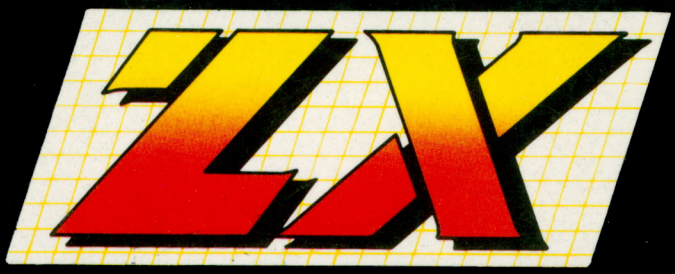
biblioteca




# PROGRAMACION PASO A PASO CON EL SPECTRUM

Philip Williams

LeG PARANINFO SA

# Programación paso a paso con el SPECTRUM

Philip Williams

# Programación paso a paso con el SPECTRUM

PARANINFO S.A. y ZX

MADRID, 1985

Traducido por:
JUAN ARENCIBIA

© Philip Williams

Reservados los derechos para todos los países. Ninguna parte de esta publicación, incluido el diseño de la cubierta, puede ser reproducido, almacenado o transmitido de ninguna forma, ni por ningún medio, sea éste electrónico, químico, mecánico, electro-óptico, grabación, fotocopia o cualquier otro, sin la previa autorización escrita por parte de la Editorial

IMPRESO EN ESPAÑA
PRINTED IN SPAIN
por: Paraninfo, S.A., en colaboración
con ZX

Depósito legal: M. 4.380.—1985
ISBN: 84-283-1373-3

PARANINFO S.A. Magallanes, 25 - 28015 MADRID (3-3491)

Artes Gráficas Benzal, S. A. - Virtudes, 7 - 28010 Madrid

# INDICE

**INTRODUCCION** EL PORQUE DE ESTE LIBRO ..... 1
¿Para quién? ..... 1
La otra cara de la moneda ..... 2
Tirarse al agua ..... 2
Importante ..... 2

**CAPITULO 1** SIGUIENDO LOS PASOS DE UNA BUENA PROGRAMACION ..... 3
Programación significa listar ..... 3
Un ejemplo simple ..... 4
Buenas y malas técnicas de programación ..... 6
La documentación es muy importante ..... 8
Reglas para programar bien ..... 9
¿Qué es la programación estructurada? ..... 9
Programación lineal ..... 10
Subrutinas anidadas ..... 12
Pros y contras ..... 14
Nunca lo tuvo tan fácil ..... 15

**CAPITULO 2** HERRAMIENTAS DEL BASIC ..... 17
Cogiendo el gusto a la programación ..... 17
Introducción al BASIC ..... 17
Reglas generales del Basic ..... 18
Enumeración de líneas ..... 18
Espaciado ..... 19
Numerando con intervalos de 10 ..... 20
¿Cuántas instrucciones por línea? ..... 20
Claridad y legibilidad ..... 21
El ordenador necesita información con la que trabajar 22
Primeros pasos ..... 22
Almacenamiento intermitente de un programa en cinta 23
Algunos comandos en BASIC ..... 23
INPUT (Entrada) y PRINT (Salida) ..... 23
Las letras pueden almacenar información ..... 24
Las Variables son muy importantes ..... 24
Operadores aritméticos + y — ..... 26
El ordenador tomando decisiones ..... 27
Las funciones del ,,;: y "" ..... 28

El punto y coma ..... 29
Instrucción REM ..... 31
Ejercicios ..... 32
Resolver ..... 33

**CAPITULO 3** EL ORGANIGRAMA ES UNA BUENA AYUDA ..... 35
El concepto de organigrama ..... 35
Explicación de los símbolos del diagrama ..... 37
Más símbolos del organigrama ..... 38
Pros y contras de los organigramas ..... 39
Instrucción IF-THEN ..... 41
El comando GO TO ..... 43
La combinación de GO SUB-RETURN ..... 44

**CAPITULO 4** PREPARANDO LOS PROGRAMAS ..... 51
51 Primera tarea: Conversiones de medidas y longitudes
Resumen de los pasos a seguir ..... 51
1er Paso: Definir detalladamente el trabajo ..... 52
2.º Paso: Diseñar el formato de salida ..... 55
3er Paso: Diseñar el formato de entrada ..... 58
El menú del trabajo ..... 59
4.º Paso: Trabaje con los posibles módulos de programación ..... 61
Diagrama de flujo en palabras ..... 66

**CAPITULO 5** CODIFICANDO EL PROGRAMA ..... 67
Observaciones generales ..... 67
Primero el programa principal ..... 68
El módulo principal de control ..... 68
Rutina de finalización ..... 70
Rutina de inicialización ..... 71
Rutinas de preparación: conversión de pesos y medidas 73
Cuatro rutinas de 3er nivel ..... 76
Trabajando con números ..... 77
Las rutinas de salida ..... 78
Las rutinas que generan las opciones ..... 80
Ahora le toca a usted ..... 82
Cometiendo errores ..... 83
¿Y ahora qué? ..... 83
Mantenga el desarrollo lineal ..... 84

**CAPITULO 6** OTRAS HERRAMIENTAS DEL BASIC ..... 85
Variables alfanuméricas ..... 85
AND, OR y NOT ..... 86
El símbolo + como operador alfanumérico ..... 87

CLEAR ..... 87
CLS ..... 88
DATA y READ ..... 88
RESTORE ..... 90
DEF FN ..... 90
Importante ..... 92
DIM ..... 92
Importante ..... 95
STOP ..... 95

**CAPITULO 7** MAS HERRAMIENTAS DEL BASIC ..... 97
FOR-TO-NEXT STEP ..... 97
Importante ..... 100
INKEY$ ..... 100
El comando LET ..... 102
STR$ (X) ..... 102
VAL (X$) ..... 103
Utilice su imaginación ..... 104

**APENDICE A** ALGUNAS FUNCIONES MATEMATICAS CON FORMATO DEF FN ..... 105
Enteros y fracciones ..... 105
Horas y grados, minutos y segundos a horas decimales o grados ..... 106
Función de redondeo ..... 106
Trigonometría ..... 107

**APENDICE B** UNA SOLUCION AL EJERCICIO DEL CAPITULO 2 109

# INTRODUCCION:
# El porqué de este libro

Esta introducción nos orientará acerca del objetivo del libro y su posterior uso.

Damos por supuesto que usted habrá desempaquetado su computadora, conectado todos los cables y hasta habrá ejecutado algún que otro programa. A lo mejor ha intentado hacer programas serios. Pero se encuentra perdido. Aunque entienda el comienzo de este libro, puede que el resto se le haga muy pesado. Conocemos esta sensación de miedo: en la «edad antigua» de la informática, adquirimos uno de los primeros micros, pero tuvimos que descifrarlo entre nosotros con uno de los peores manuales que conocimos... Pero créalo, la constancia ganó después de todo. O sea, que no se sienta inseguro solamente porque no sabe cómo trabajar con su ordenador, ni cómo hacer ese programa que tiene en mente durante tanto tiempo, y que le hicieron creer era muy fácil. Como todas las artes, el arte de programar no se adquiere de un día para otro. Lleva tiempo y dedicación, pero no es difícil. También hace falta una buena guía y ese es el objetivo de este libro.

## ¿PARA QUIEN?

Este libro ha sido escrito, para el novato de la programación, en un lenguaje llano y no muy complicado. Se intentará evitar ciertas expresiones de más alto nivel cuando se trate de programar.

Por otra parte, este libro le servirá de ayuda si ya sabe algo de programación, esto es, si ya ha escrito algún que otro programa y los ha ejecutado. El que un programa ejecute no significa necesariamente que sea un buen programa. Puede ser un programa muy descuidado, por diversas razones que comentaremos más adelante. La mayor parte de este libro trata de técnicas de programación que le darán una pequeña ventaja con cualquier programa que intente y un final feliz cuando lo realice.

## La otra cara de la moneda

Los ordenadores personales requieren una participación plena del individuo. La ventaja del auto-aprender reside en la diversión, en la creatividad y en la gran satisfacción que se puede obtener de ello. Pero la otra cara de la moneda es que si uno no aprende bien, quedará al margen de la inapreciable técnica que ha sido desarrollada por los maestros de este arte. Esta observación general se puede aplicar al campo de la programación. Específicamente, los programadores de BASIC que han aprendido por su cuenta tendrán que olvidar —por desgracia— muchos de los hábitos indeseables antes de aprender a aplicar sus conocimientos a otros lenguajes de programación.

## Tirarse al agua

A menudo, a quienes desean aprender a programar se les aconseja que se «tiren al agua y naden». No es este el objeto de este libro. Para olvidar los malos hábitos contraídos de la experiencia propia y para adquirir las técnicas de programación que son reconocidas como correctas, nuestro consejo es seguir el libro y trabajarlo hasta que se le indique lo contrario. Los capítulos están ordenados secuencialmente, y hay ejercicios en alguno de ellos. Haga un esfuerzo real en estos ejercicios para conseguir el máximo beneficio.

## Importante

Este libro no prentende sustituir al libro de introducción y manual del ZX Spectrum. Es importante, por lo menos, aprenderse el libro de introducción para familiarizarse con el teclado.

## CAPITULO 1:

# Siguiendo los pasos de una buena programación

Este capítulo subraya la idea de lo que está considerado como una mala programación y lo que generalmente se considera como buena programación. Pero antes de entrar en detalles nos debemos hacer una pregunta: «¿Qué es la programación?» No es una pregunta tonta; al contrario, muchas personas, incluyendo a usted, tienen una idea equivocada de lo que es la programación.

### PROGRAMACION SIGNIFICA LISTAR

Primero, dejemos claro que la noción de programa puede sugerir múltiples ideas. Hablamos de programas de televisión y de programas de radio, programamos un día en el campo, un programa político, el programa de un concierto en la ópera y en general de cualquier tipo de programa. La diferencia entre todos estos tipos de programas son obvias pero todas tienen una cosa en común. La esencia de un programa es una lista de palabras que nos envían en mensaje y un significado. Pero si nos referimos al mundo de los ordenadores, un programa es una lista de instrucciones.

Para dejarlo claro, pongamos un ejemplo de la vida real. Supongamos que usted nunca ha conducido un coche, pero quiere aprender. El paso más lógico sería ir a una autoescuela y que le asignen un profesor. Este le sentará en el asiento del conductor y le explicará las funciones de todos los botones y palancas del coche. ¿El hecho de entender bien el funcionamiento de todos los botones le permitirá conducir bien el coche? Seguramente que no. Por lo que el paso siguiente sería que su profesor le enseñara la secuencia lógica para poner en marcha el coche y conducirlo. Por ejemplo: sólo hay

una forma de que el coche se ponga en marcha y es arrancar el motor con la llave de contacto mientras se acelera un poco y se suelta el embrague, etc., etc... No se puede hacer otra forma. Hay un juego de reglas a seguir en un orden determinado para conseguir que el coche se desplace del punto A al B.

Este principio también se puede aplicar a los ordenadores para asegurar su correcto funcionamiento. Después de todo, y como en el ejemplo del coche, el ordenador es sólo una máquina que no puede hacer nada por su cuenta. Necesita una serie de instrucciones en una secuencia determinada antes de que ejecute el trabajo que uno quiera.

Este es el quid de la cuestión, antes de que el ordenador las memorice, estas instrucciones tienen que ser escritas con mucho cuidado: esto es, que esté exactamente como lo quiere la máquina, deletreada correctamente y en una secuencia correcta. Si no se siguen estas reglas hay un alto porcentaje de posibilidades de que la máquina no entienda las instrucciones, o que se vuelva loca al colocar las instrucciones en una secuencia incorrecta. En el primer paso tenemos los SYNTAX ERRORS (Errores sintácticos) y en el segundo caso hablamos de LOGIC ERRORS (Errores lógicos).

## UN EJEMPLO SIMPLE

Para ilustrar esto, hagamos un programa muy simple. Teclee este programa tal y como está:

```
 5 LET B=0
10 LET A=B+1
20 LET B=A
30 PRINT A
40 GO TO 10
```

NOTA: Esperemos que ya haya pasado algún programa del manual del ordenador, de manera que sabrá como teclear el programa. Si no lo ha hecho todavía, siga estos consejos. Encienda su máquina (y naturalmente su TV o monitor, conectado al ordenador). Entonces teclee 1Ø LET A (espacio) = (espacio) B (espacio) + (espacio) 1 y luego pulse la tecla ENTER. La línea de programa (ya que eso es lo que es) está ahora segura en la memoria del ordenador. Haga lo mismo con las líneas 2Ø, 3Ø y 4Ø.

El paso siguiente es ejecutar el programa. Se hace con la tecla R, que tiene la palabra RUN y luego se pulsa ENTER.

Lo que se ve ahora es cómo, a una velocidad increíble, los números van ocupando la pantalla desde el 1 en adelante. Entonces el ordenador le preguntará ¿scroll? Si quiere continuar rellenando la pantalla, basta con pulsar la tecla ENTER.

Puede seguir así indefinidamente y cuando menos lo espere el programa habrá realizado miles de cuentas (porque eso es lo que está haciendo, contar). Y ¿si no tuviera una interrupción «scroll?», seguiría rellenando sin parar.

Otro truco. El programa sigue en la memoria de su ZX Spectrum, y puede borrar una línea simplemente tecleando el número de línea a borrar y pulsando ENTER, o puede teclear ese número de línea con la nueva función a realizar, por ejemplo: 4Ø GO TO 1ØØ y ejecutar de nuevo el programa. De ninguna manera ejecutará. El ordenador colocará sólo un 1 en la pantalla y se acabó. Este es un típico error de lógica. La lógica del programa está mal, porque el ordenador se dirige hacia una línea de programa que no existe, línea número 1ØØ. En otras palabras, la línea 1ØØ no ha sido definida.

Pero hay errores de lógica que no están tan claros. Este es el caso de que un programa parezca ejecutar correctamente pero responde con las respuestas erróneas al problema planteado. Volvamos a nuestro ejemplo. Teclee 4Ø GO TO 2Ø, y ejecute el programa. Lo que ahora ve son una serie de 1 a través de la pantalla. Por supuesto que no era el objetivo del programa. El error lógico está en que el programador ha dirigido al ordenador hacia la sentencia equivocada. Como consecuencia de esto se repite la sentencia B = A, en vez de ejecutar la suma (línea 1Ø LET A = B + 1).

Uno de los peores errores en que puede incurrir un programador está en el bucle sin fin. Pruebe esto: (teclee NEW y ENTER primero para borrar programas anteriores de la memoria) 1Ø GO TO 1Ø. Sintácticamente, este programa está correcto: cada letra y dígito está escrito correctamente. Pero cuando se ejecute efectuará un trabajo sin sentido para nosotros, y se meterá en un bucle sin fin.

En este programa la línea 1Ø GO TO 1Ø hace refe-

rencia constante a la línea 1Ø, provocando un bucle que no tiene posibilidad de finalizar, no se le puede para ni con la tecla BREAK. La única forma de acabar es desenchufando el ordenador.

El programa de la página también es un bucle sin fin, pero efectúa un trabajo dentro de ese bucle.

Ahora ya tiene una idea de lo que es un programa. Naturalmente, lo explicaremos con más detenimiento en los capítulos consecutivos.

## BUENAS Y MALAS TECNICAS DE PROGRAMACION

¿Qué es lo que llamamos buena y mala programación? En vez de teorizar acerca de todas sus posibles respuestas, lo mejor sería ver un par de ejemplos.

NOTA: Antes de entrar en detalles, queremos constatar que ambos problemas se han exagerado en apariencia y que los dos pueden resolver la misma tarea de una manera más simple.

Ambos programas nos darán los mismos resultados cuando les demos los datos con los que tienen que trabajar. Por lo que, quizá, su primera impresión será que los dos programas son buenos porque dan los mismos resultados. Incluso pensará que el más corto es mejor porque es más corto. En estos momentos lo que más importa es la forma de plantear y describir el problema y no los resultados. Por esa razón no siempre un programa corto será siempre un buen programa.

Prog. A,

```
 5 CLS
 7 LET x=60
10 INPUT h
11 INPUT m
12 INPUT s
20 LET s=s/x
30 LET m=m+s
40 LET m=m/x
50 LET h=h+m
60 PRINT h
```

Prog. B,

```
100 REM Convierte horas-grados
110:
120 REM este programa convierte
130 REM horas, minutos y seg.
140 REM a horas decimales
150:
160 REM autor Edward P Barton
170 REM escrito el 09-02-1984
180:
200:
210 REM entrada de datos
220:
225 REM almacena horas en hr
230 INPUT "horas  ";hr
235:
240 REM almacena minutos en mn
245 INPUT "minutos ";mn
250:
255 REM almacena segundos en sc
260 INPUT "segundos ";sc
265:
270 REM rutina de calculos
275:
280 REM convertir segundos a
282 REM decimales
285 LET sc=sc/60
290:
300 REM sumar segundos
303 REM decimales
305 REM a los minutos
310 LET mn=mn+sc
315:
320 REM convertir minutos a
323 REM min. decimales
325 LET mn=mn/60
330:
335 REM sumar horas decimales
340 REM a los minutos
350 LET hr=hr+mn
355:
360 REM imprimir el resultado
365:
370 PRINT "convertido a decimal
es:"
375 PRINT : REM linea en blanco
380 PRINT hr;"  horas"
```

Ahora eche una ojeada a ambos ejemplos. No tiene que ser un programador experimentado para descubrir que aunque el programa A es sensiblemente más corto, es más difícil de entender que el ejemplo más largo B. Ahora supongamos que usted es un programador con

experiencia. Esto no garantiza que sepa lo que el programa A tiene que hacer. Al contrario, si no está familiarizado con el tipo de problema que resuelve el programa, éste seguirá siendo un misterio. Por otro lado, el programa B le indica a la primera el objetivo del programa. Es más, el planteamiento es elegante, con comentarios en casi todas las líneas (por las sentencias denominadas REM), y las líneas se han dejado a doble espacio para mejorar su legibilidad. El programa B tiene un toque de profesionalidad. El programa A, aunque hace bien su cometido, es un ejemplo de una programación pobre.

## LA DOCUMENTACION ES MUY IMPORTANTE

Veamos otro ejemplo, que no tiene mucho que ver con la programación, pero nos puede dar una idea de lo que queremos conseguir. Pida a un mecánico experimentado que repare un modelo de coche que nunca ha visto en su vida. Sin duda alguna entenderá los pasos básicos para reparar un coche, pero cuando entre en detalles según el modelo se encontrará perdido. Por lo que el paso siguiente será comunicarse con el fabricante o el distribuidor para obtener la documentación técnica necesaria para repararlo.

Pero eso no será suficiente. Para poder trabajar bien, necesitará toda la documentación necesaria y que esta sea completa, clara y concisa. Es más, tiene que ser fácil de seguir. Ya que si en cada párrafo se tiene que preguntar lo que quería decir tal comentario u otra cosa, nos encontraremos con un mecánico desesperado.

Volvamos a la programación. Imagínese que le han prestado o se ha comprado un listado de un programa con el que quiere experimentar. Le aseguramos que se sentirá perdido cuando le presenten una larga lista de palabras, en vez de un programa bien documentado, y fácil de seguir. Y le sorprenderá saber que más de un programador profesional son incapaces de descifrar aquellos programas que hicieron algún tiempo atrás.

Volvamos a los programas A y B. En este momento podrá comprender lo que queríamos dejar sentado. Pero después de un tiempo, por ejemplo un año, vuelva a ver dicho programa, lo más seguro es que no lo entienda. Por otro lado, después de cierto tiempo sin ojear el programa B, le será más fácil comprender lo que se quería

conseguir con él, y todo gracias a la documentación y la forma de presentarlo. Es un programa bien estructurado.

## REGLAS PARA PROGRAMAR BIEN

Hemos planteado la analogía del mecánico de coches para demostrar lo importante que es el poseer buena documentación. Pero no es tan efectivo como la forma de escribir un programa. Por eso es sumamente importante conocer algunas de làs reglas básicas cuando haga cualquier tipo de trabajo, ya sea reparar un coche o programar ordenadores. En cuanto a la programación todo se reduce a lo siguiente:

- El trabajo de programación debe estar bien preparado. Por lo menos habrá que especificar lo más preciso posible lo que se espera conseguir con el programa.
- El programa ha de estar cuidadosamente estructurado, esto es, se tendrá que hacer por bloques y cada bloque ejecutar una tarea específica.
- El programa ha de tener un comentario de cada línea.
- En términos generales, el programa ha de ser fácil de seguir. Gracias a esto y a la documentación, se le podrá entender, modificar o incluso actualizarlo caso de que sea necesario.

Es verdad que la mayoría de los programas escritos en BASIC no tienen las cualidades que acabamos de describir. Por lo que la mayoría de ellos son un ejemplo de programas pobres, aunque su cometido lo ejecuten a la perfección.

Lo que acabamos de definir como reglas, hacen que un programa sea un «buen» programa. Son algunas de las reglas que comprenden el arte de la programación estructurada. Entraremos en detalles más adelante, pero asegúrese que saca el máximo beneficio de estas reglas, ya que la programación le será más fácil de esta forma.

## ¿QUE ES LA PROGRAMACION ESTRUCTURADA?

Tal vez sería más sencillo definir lo que no es la programación estructurada. Pero digamos que es la forma de programar que describe los casos de prueba/error.

Esto está basado en la aceptación general de que programar es sentarse frente al ordenador y teclear el programa, comprobando lo que sucede. Cada vez que cometa un error la máquina se lo dirá y entonces lo rectificará. Y así, línea a línea, escribirá el programa, cambiando y mejorando las sentencias cada vez que quiera. Desde luego que este procedimiento mejorará la creatividad de las personas, que es de lo que trata cuando trabajamos con ordenadores personales, haciendo a estos más divertidos y entretenidos de utilizar. Esto hará que los programas funcionen, en una palabra, que hagan el trabajo. Pero ¿serán eficaces esos programas? En la mayoría de los casos no. Es como volar desde Sydney hasta Roma, haciendo escalas en Nueva York, Johanesburgo, Reihyavik, Nueva Delhi y Buenos Aires. La mayoría de las veces, estos programas saltan por el programa sin orden ni concierto. Sólo los autores entienden sus propias creaciones... al principio. Ya que la experiencia nos dice que después de un tiempo esos autores pierden la pista a sus propios programas. Es como atravesar un laberinto; al principio es muy divertido pero más adelante se convierte en una frustación. En el párrafo anterior hablamos de programas que parecen puzzles, que no tienen documentación o, lo que es peor, que empiezan en algún lugar del listado, luego cambian de operación y van a un bloque de líneas al principio, luego de repente se dirigen al centro del programa para volver al principio, más adelante se ejecutan las últimas veinte líneas... un completo desorden.

Al principio este tipo de programa puede ser divertido pero más tarde se convierte en desastroso. Es frustante intentar descifrar lo que actualmente se ha hecho para resolver semejante problema o incluso seguir paso a paso la comprobación de un resultado o dándose cuenta que todavía no sabe lo que puso en la línea 1.321, o el objeto de una sección completa empezando en la línea 4Ø444. Intentamos descifrar un programa realizado hacía un tiempo y nos dimos cuenta que una sección entera no tenía sentido. Ninguna parte del programa le hacía referencia.

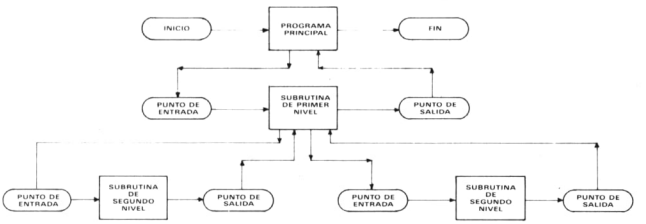
## PROGRAMACION LINEAL

Esencialmente la idea es muy simple: un programa comienza al principio y acaba al final. Pero hay más que

esto, de lo contrario no sería un apartado tan importante en el mundo de la informática. El principio de la programación lineal lo podrá entender cuando haya hecho algún programa de acuerdo con sus reglas. Por eso no nos extenderemos demasiado ahora, el capítulo siguiente lo hará con más detalles.

Dejemos sentado que la idea de programación lineal hay que mantenerla a través de todo el programa. Según explicamos anteriormente, un buen programa está bien estructurado en cuanto se compone de varios bloques, cada bloque ejecutando tareas distintas y específicas. Por lo que los bloques se pueden considerar como pequeños programas separados, sub programa o, como prefieren denominarlo los programadores del BASIC, subrutinas.

Una subrutina comienza al principio y acaba al final de acuerdo con el principio de la programación lineal. Esto es lo más importante, la primera línea marca el comienzo de los datos que han de ser procesados en la subrutina, y la última línea marca el punto de salida de dicha subrutina. La regla general en programación estructurada es que la subrutina no puede tener más de un punto de entrada y no más de un punto de salida. Para dejarlo claro vamos a compararlo con una fábrica, donde normalmente hay un área de materiales en bruto y en otra el producto final que se comercializa.




Para volver a la subrutina, el punto de entrada recibe los datos que serán procesados y el punto de salida manda los resultados adecuados. En la jerga informática, al punto de entrada se le denomina INPUT y al punto de salida OUTPUT o RETURN del OUTPUT.

Más adelante trataremos profundamente acerca de las subrutinas, pero por ahora conviene acabar con la idea muy generalizada entre los programadores de que las subrutinas han de ser lo más cortas posible, preferible-

mente no más largas de un folio. Si el programador se encuentra en la necesidad de utilizar una subrutina dentro de otra, lo puede hacer perfectamente, particularmente si mantiene la idea principal de que cada subrutina ha de hacer un trabajo en particular.

## SUBRUTINAS ANIDADAS

Hemos utilizado el término de subrutina anidada sin haberlo explicado anteriormente. Ahora lo definiremos. Otra regla más a tener en cuenta en lo que se refiere a la programación estructurada es que las subrutinas se ponen en el programa en un determinado orden. Para ilustrar esto, observe el diagrama de esta página. Explica la programación estructurada hasta en sus más pequeños detalles. Como se ve aquí, al principio encontramos el cuerpo principal del programa —los programdores lo denominan línea principal, ya que ahí es donde empieza y donde termina—. De todos modos utiliza una subrutina a la cual se le asigna una parte del proceso. Como vemos el diagrama se bifurca en ramas desde el tronco y entra en las subrutinas anidadas, denominadas subrutinas de primer nivel y al finalizar vuelven al programa principal. Las subrutinas de primer nivel utilizan dos subrutinas que, como consecuencia de ello, se denominan subrutinas de segundo nivel. Como podemos comprobar estas están debajo de las del primer nivel.

En la programación real no podemos situar una subrutina al lado de la otra, como está en el diagrama. La manera de situar un programa con una subrutina de primer nivel y dos subrutinas de segundo nivel, sería la siguiente:

— Rutina principal.
— Subrutina de primer nivel.
— Subrutina de segundo nivel.
— Subrutina de segundo nivel.

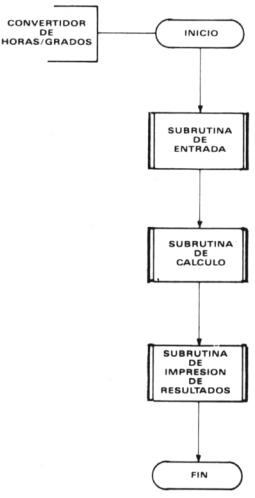
Para poner un ejemplo real, examinemos el programa que convierte las horas en grados (programa B), pero ahora escrito de acuerdo con lo explicado acerca de la programación entructurada. Ver programa.

NOTA: Si algunos lectores lo encuentran como un ejemplo muy exagerado, de esto, desde luego no se

equivocan. Se hizo así por el bien de todos y por la claridad que ello conlleva.

Ver también el diagrama que va con el programa. Este se denomina diagrama de flujo u organigrama, ya que nos dice en qué sentido se desarrollará nuestro programa. Los diagramas de flujo son una parte de la programación que ayuda bastante, pero esto lo veremos con más detalles en el próximo capítulo.

```
100 REM ******************
110 REM *                *
120 REM *   convierte    *
130 REM * horas y grados *
135 REM *                *
140 REM ******************
150:
160 REM este programa convierte
161 REM horas minutos y seg.
162 REM en horas decimales
163 REM o grados minutos
164 REM y segundos en
165 REM grados decimales
170:
180:
190 REM autor - edward barton
200 REM escrito 09-02-1984
210:
220 CLS : REM limpia la pantall
a
230:
240 REM ======================
250 REM programa principal
260 GO SUB 320
270 GO SUB 420
280 GO SUB 510
290 GO TO 600: REM fin programa
295:
300 REM ======================
310:
320 REM subrutina de entrada
330:
335 REM almacena horas en hr
340 INPUT "horas   ";hr
345 REM almacena minutos en mn
350 INPUT "minutos ";mn
355 REM almacena segundos en sc
360 INPUT "segundos ";sc
370 RETURN
375:
380 REM ======================
390:
400 REM rutina de calculo
410:
415 REM convierte segundos
416 REM a seg. decimales
420 LET sc=sc/60
425 REM sumar segundos
426 REM a los minutos
```




```
430 LET mn=mn+sc
435 REM convertir minutos a
436 REM min. decimales
440 LET mn=mn/60
445 REM sumar minutos decimales
446 REM a las horas
450 LET hr=hr+mn
455 REM vuelve al programa
456 REM principal
460 RETURN
470:
480 REM ======================
485:
490 REM rutina de impresion
500:
510 PRINT "convertido a decimal
es:"
520 PRINT hr;"   horas"
525:
530 REM vuelve a la rutina
531 REM principal
535: RETURN
540 REM ======================
550:
600 PRINT : PRINT : PRINT
610 PRINT "fin del programa"
```

## PROS Y CONTRAS

Nuestro deber es mencionar el gran debate en lo que se refiere a la programación estructurada, en cuanto a que es una desventaja y más si se utiliza un lenguaje como el BASIC. El argumento más escuchado contra este método de programación es que limita totalmente la creatividad. Pues bien, el planteamiento tipo pregunta/respuesta tiene unos aspectos muy acertados, pero para nuestra firme convicción estos no son mejores que las desventajas expuestas en párrafos anteriores. Es comprensible que aquellas personas que hayan disfrutado de la libertad de utilizar el plateamiento del tipo pregunta/respuesta se sentirán incómodos ante las reglas de la programación estructurada. Esta incomodidad no está justificada —todo depende de hasta dónde se quiera llegar con las reglas—. Este libro no le va a obligar a aceptar esas reglas tan estrictas. Estamos seguros de que se puede establecer un término medio entre la libertad máxima y la total restricción de unas reglas dogmáticas. Diseñar un programa puede ser un proceso altamente creativo y puede serlo dentro de los paráme-

tros establecidos por la programación estructurada como probarán los siguientes capítulos.

Otro argumento en contra de la programación estructurada es de naturaleza técnica. Las líneas de comentarios ocupan sitio en la memoria del ordenador y frenan la ejecución del programa. Este argumento es una reliquia de días pasados cuando los ordenadores personales tenían un espacio de memoria muy limitado, por lo que, todo lo que se consideraba como superfluo se eliminaba. Lo más superfluo que había eran las sentencias REM. De aquí el hábito de no escribirlas en los programas.

En aquellos días los programadores tenían mucho cuidado en cuanto a la velocidad de ejecución de un programa. Si se podía mejorar en unas pocas décimas de segundo, uno se preocupaba de conseguirlo. Por ejemplo, rápidamente se dieron cuenta de que en BASIC, una subrutina se busca desde la primera sentencia de un programa en adelante. Por esta razón muchos programadores situaban las subrutinas lo más arriba posible dentro de un programa para obtener esas décimas extra de segundo comenzando el programa al final. De allí vienen esos malos hábitos de la programación.

## NUNCA LO TUVO TAN FACIL

Los días en que la memoria estaba limitada han acabado. Su ZX Spectrum tiene cinco veces más memoria que aquellos ordenadores de antaño. ¡Nunca lo ha tenido tan fácil! Le podemos asegurar que el programa ha de ser larguísimo para llenar todo ese espacio. Por esta razón las líneas de comentarios ya no son un problema. Tampoco importa dónde vayan colocadas las subrutinas en lo que se refiere a la velocidad de ejecución. He aquí la confirmación de la programación estructurada. ¡Adelante con ella!

## CAPITULO 2:

# Herramientas del BASIC

### COGIENDO EL GUSTO A LA PROGRAMACION

Cuando aprendemos un oficio, por ejemplo el de carpintero, lo primero que hacemos es familiarizarnos con las herramientas de las que disponemos para trabajar. Tenemos el martillo, los alicates, la sierra y los clavos. Habrá que observar cómo un carpintero trabaja en su oficio. Algunas veces será fácil comprender lo que quiere hacer, otras tendrá que preguntar para que le explique lo que está haciendo y por qué. Esta es una forma de aprender el oficio.

Esta es la forma en que este libro pretende introducirlo en la programación. En este capítulo subrayaremos algunas de las herramientas básicas de este lenguaje.

### INTRODUCCION AL BASIC

La palabra BASIC no sólo significa que es un lenguaje muy básico. En realidad BASIC significa Beginners All-purpose Symbolic Instruccion Code (Lenguale Simbólico para principiantes) y fue desarrollado por un grupo de trabajo de la Universidad de Darmouth (California) en la década de los 60, con el propósito de servir como lenguaje de programación para principiantes. Probablemente para sorpresa de sus inventores, el BASIC se ha convertido en el lenguaje de programación más popular en la actualidad, y aunque parece estar hecho para principiantes ha llegado a convertirse en un lenguaje para profesionales, que podemos hallar en cualquier tipo de micro-ordenador. No tan bueno ha sido el desarrollo de múltiples versiones o «dialectos» de este lenguaje. El BASIC de su ZX Spectrum es uno de esos dialectos. Esto implica que cuando haya dominado una de las versiones particulares del BASIC, todavía tendrá

que hacer algunos esfuerzos si quiere trabajar con otras versiones del mismo lenguaje. De todos modos, todos los BASIC tienen un juego estándar de instrucciones en común, por lo que el pasar de una versión a otra no es difícil.

## REGLAS GENERALES PARA EL BASIC

BASIC es un lenguaje y, como ya sabrá, cada lenguaje tiene sus reglas. A esto lo denominamos sintaxis. Los idiomas humanos son más flexibles aplicando sus reglas, mientras que los lenguajes de programación apenas son flexibles. Tienen que escribirse de acuerdo a unas reglas. Después de todo, aúnque muchas personas les atribuyen unos poderes mágicos a los ordenadores, estos no son más que unas máquinas que no saben pensar por sí solas. Por esta razón los ordenadores no reconocerán las instrucciones que no estén escritas precisamente de acuerdo con unas reglas.

Con su ZX Spectrum, no tendrá ese problema debido a que los comandos e instrucciones son activados al pulsar una sola tecla. En comparación con otros ordenadores no tiene que teclear el comando letra a letra, por lo que nunca tendrá un error sintáctico causado por la mala escritura de la palabra o comando.

## ENUMERACION DE LINEAS

Un programa en BASIC se escribe en líneas, conteniendo estas las instrucciones que ejecutarán en el programa. Cada línea tiene que estar numerada y un conjunto de líneas ha de numerarse en un orden creciente. No importa cómo las numere mientras que la línea anterior tenga un valor inferior a la línea siguiente. Por ejemplo, la siguiente secuencia de líneas de un programa en BASIC está numerada correctamente:

```
  1 LET B=1
 10 LET A=B+1
 99 PRINT A
111 GO TO 10
```

Pero también se puede escribir:

```
1 LET B=1
2 LET A=B+1
3 PRINT A
4 GO TO 2
```

E incluso:

```
1000 LET B=1
2000 LET A=B+1
3000 PRINT A
4000 GO TO 2000
```

o cualquier otra secuencia de numeración mientras mantenga la regla de incrementar el número.

IMPORTANTE: El BASIC se lo pone fácil a la hora de añadir, quitar o sustituir líneas de programa. Por ejemplo, si acaba de teclear el programa anterior y quiere insertar la línea 15ØØ, sólo basta con poner el número 15ØØ con su correspondiente instrucción. Cuando ejecute el programa, el ordenador automáticamente colocará la nueva línea de programa en su lugar correspondiente. Esto se aplica a la hora de añadir o insertar líneas. Si se quiere suprimir una línea de programa, bastará con teclear el número y pulsar la tecla ENTER. La línea entonces desaparecerá del programa. Suprimir, añadir o alterar una línea es lo que se conoce con el nombre de editar. Comprobará que los desplazamientos del cursor son de gran ayuda.

## ESPACIADO

Algunos lenguajes de programación tienen unas reglas estrictas en lo que se refiere al espaciado entre las palabras de una instrucción o comandos. En la mayoría de los casos, el BASIC es flexible, aunque el BASIC de su ZX Spectrum incluye automáticamente el espacio antes y después de una palabra clave o comando. Vea el ejemplo siguiente:

```
1 LET B=1
2 LET A=B+1
3 PRINT A
4 GO TO 2
```

Como puede comprobar, las líneas 1 y 2 están un poco comprimidas. Su ordenador también aceptará lo siguiente:

```
1 LET B = 1
2 LET A = B + 1
```

lo que, por supuesto, no es nada práctico.

De todos modos, el mejor consejo es que mantenga la legibilidad de sus programas, evitando mientras no sea necesario el comprimir las líneas de los programas.

Cuando haga algo que va en contra de las reglas (o de la sintaxis) del BASIC, su ordenador le avisará de inmediato mediante un mensaje de error que aparecerá en la pantalla de su televisor o monitor. Podrá corregir este error, llamado comúnmente error de sintaxis, inmediatamente, por ejemplo tecleando la línea errónea de nuevo. Esta cualidad especial del BASIC (visualizar los errores inmediatamente y haciendo que el programador los corrija al momento) hace que este lenguaje se denomine interactivo.

## NUMERANDO A INTERVALOS DE DIEZ

Hay un acuerdo general entre los programadores para que las líneas de programas se numeren de diez en diez. La secuencia 1, 2, 3, 4, etc., no le permitirá insertar líneas extras en el caso de que se necesiten (y esa necesidad aparece cuando uno menos se lo espera). La secuencia 1Ø, 2Ø, 3Ø, etc., ofrece la posibilidad de poder insertar líneas extras.

Es también bueno para programar —por lo menos en el caso de programas extensos— comenzar por la línea 1ØØ. Cuando el programa se haya escrito completamente, podrá insertar cuantos comentarios quiera antes de la línea 1ØØ.

## ¿CUANTAS INSTRUCCIONES POR LINEA?

El ZX Spectrum permite escribir más de 9Ø caracteres por línea. Un carácter es el nombre general que se

le da a un dígito, letra, punto, coma, gráfico o lo que tenga en el teclado. Hasta un espacio en blanco se considera como carácter.

Pero en beneficio de la claridad y legibilidad, sugerimos que las líneas de programación sean cortas. En general, lo mejor es que una línea de programa no sobrepase una línea en la pantalla. Una línea de programa en la pantalla del televisor contiene 32 caracteres; procure mantenerse dentro de estos límites.

## CLARIDAD Y LEGIBILIDAD

¿Se acuerda de las reglas de la programación estructurada? Claridad y legibilidad son algunas de ellas. Esto se reduce a tres comentarios:

1. Teclee espacios entre instrucciones. Ya lo hemos discutido en el párrafo del espaciado.

2. Utilice líneas de programas abiertas de vez en cuando. El BASIC de ZX Spectrum facilita esta operación. Vea el ejemplo:

```
10 INPUT A
15:
20 PRINT A
```

Los dos puntos utilizados en la línea 15 es una línea abierta.

3. Use las líneas de comentarios para especificar las acciones que toma en cada paso. Otra vez el BASIC facilita esta labor con la instrucción REM. REM significa «REMark» (del inglés comentarios). Lo puede colocar en cualquier parte del programa mientras no interrumpa el flujo del programa.

Ejemplo:

```
 5 REM este programa calcula..
10 INPUT A:  REM entrada de
                 datos
15:
20 PRINT A:  REM salida de
                 datos
```

## EL ORDENADOR NECESITA INFORMACION CON LA QUE TRABAJAR

Antes de que entremos en detalles sobre las palabras clave del BASIC, discutiremos dos palabras que usted habrá oído mencionar mucho antes de pensar en comprarse un ordenador personal, «input» (entrada) y «output» (salida). Son reminiscencias del mundo de los amplificadores y tocadiscos, pero reminiscencias o no, se han convertido en palabras utilizadas en el lenguaje diario.

Para ser claros, «input» es el proceso por el cual damos información al ordenador. Cuando le damos algo para trabajar, lo aceptará y lo procesará, obteniéndose así unos resultados que denominamos «output».

El «input» (la entrada de datos) puede ser de varias maneras. A través del cassette o en discos magnético, por ejemplo. Pero en nuestro caso también acepta datos de entrada por el teclado.

El «output» (salida de datos) también puede hacerse en más de una forma: comunmente se utiliza la pantalla del TV o el monitor, o por impresora, cuando se requiera.

Por el momento nos preocuparemos de la información vía teclado, o entrada manual, y los resultados los veremos en pantalla, o salida por pantalla.

## PRIMEROS PASOS

Antes de seguir, una pregunta confidencial:

¿Ha estudiado y hecho todos los ejercicios en el manual de Introducción del Spectrum?

Si su respuesta es negativa, le sugerimos que los haga antes de seguir adelante. Es como la primera lección de conducir, cuando el profesor le dice «aquí está el freno, allí el embraque, este es el contacto... » etc. Esta introducción es absolutamente necesaria, si todavía no sabe utilizar el cursor, o cómo poner las palabras clave (comandos), o borrar letras, y otros caracteres, etc. De otra forma no irá mucho más lejos en este libro.

## ALMACENADO INTERMITENTE DE PROGRAMAS EN CINTA

Escribir y probar los programas no es problema de unas pocas horas. Muchas veces hacen falta días, y cuando el trabajo es largo hacen falta hasta meses.

Como consecuencia de ello tiene que almacenar el programa en cinta durante todas las etapas del proceso. Es obvio que el ZX Spectrum no se puede dejar conectado para siempre. De todos modos cuando se apaga el ordenador toda la información almacenada en ese momento se borra, por lo que al final de una jornada de programación tendrá que guardar en cinta lo que ha hecho. Al día siguiente, cargará desde la cinta lo que había desarrollado hasta entonces y continuará con el trabajo.

Guardar y cargar desde el cassette es muy fácil. En el manual del Spectrum viene muy bien detallado.

## ALGUNOS COMANDOS EN BASIC

Empezaremos con algunas instrucciones, conocidas generalmente como comandos. Con estos comandos y unos pocos símbolos estará preparado para escribir programas simples.

## INPUT (ENTRADA) Y PRINT (SALIDA)

¿Cuál es su misión? Con el comando INPUT le damos al ordenador la información para que sea procesada por los programas. El comando PRINT, visualizará los resultados por pantalla, o en combinación con otros comandos saldrá por impresora.

Encienda su ordenador y el TV o monitor, y teclee lo siguiente:

```
1Ø INPUT A
2Ø PRINT A
```

ejecute este pequeño programa (pulse RUN y ENTER).

Aparecerá un signo de interrogación parpadeante en la pantalla. Ahora teclee 23 y pulse ENTER. Comprobará que el ordenador ha escrito en la pantalla el número 23 justo debajo del que usted acaba de teclear.

¿Cómo lo ha hecho? la respuesta requiere unos pocos conocimientos más.

## LAS LETRAS PUEDEN ALMACENAR INFORMACION

¿Qué sentido tiene la A en este programa, cuando hacemos INPUT A y PRINT A? Para explicarlo veamos un ejemplo de un archivador con muchos cajones. Como puede ver el archivador almancena información. Si el propietario es una persona metódica lo más seguro es que tenga los archivadores etiquetados con un nombre, para hacer más fácil su localización. Por ejemplo, el cajón a contendrá las facturas de la casa, el cajón B, las notas de la escuela, el C, la colección de sellos, etc. La misma idea se aplica para almacenar información dentro del ordenador en un lenguaje como el BASIC. Un ordenador tiene «celdas» para almacenar la información necesaria para ser tratada en futuros programas. Estas «celdas» tienen que estar etiquetadas en la forma de una o dos letras, o una combinación dígito-letra. Por ejemplo: A, AA, A1, AB, BX, X9, etc.

Cuando escribimos el programa 1Ø INPUT A, lo que queremos decir es que la línea 1Ø del programa indique al ordenador que mire en la «celda» cuya etiqueta es A y responda con el contenido de dicha celda vía pantalla.

Tenga presente que como los archivadores, las «celdas» del ordenador mantendrán la misma etiqueta aunque tengan diferente contenido. Puede almacenar las facturas de la casa en esta «celda» pero más adelante a lo mejor quiere cambiarlo por la colección de sellos. Pero el nombre del cajón permanecerá inalterable. Lo único que varía es el contenido. Por esta razón en el mundo INFORMATICO a estas «celdas» imaginarias se las conoce por el nombre de VARIABLES.

## LAS VARIABLES SON MUY IMPORTANTES

Estos identificadores de la celda o más concretamente variables, forman una herramienta muy potente en programación. Raramente encontrará un programa sin ellas. En verdad pertenecen al bloque principal dentro de un programa. Cuando tenga mucha información almacenada en variables, podrá manipularlos y barajarlos a su

gusto. Después de todo, todo ordenador, el suyo incluido, es un manipulador de información. Compruebe este ejemplo:

```
 10 INPUT A
 20 INPUT B
 30 INPUT C
 40 INPUT D
 50 LET E=A+B+C+D
 60 PRINT E
 70 LET E=A*B*C*D
 80 PRINT E
 90 LET A=E
100 LET B=E+A
110 LET C=E+D
120 LET F=A+B+C
130 PRINT F
```

Este ejemplo le enseñará cómo toda la información o los datos, como prefieren llamarlos los programadores, son manipulados en este programa gracias a la utilización de las variables. Un programa es un conjunto de instrucciones y cada vez que le demos información a procesar mediante INPUTS, tendrá distintas salidas (OUTPUTS). Pero eso es lo que tiene que hacer un ordenador. Por ejemplo, podrá escribir un programa que convierta los pies en metros, decímetros, centímetros y milímetros. Un programa que hace semejante trabajo puede parecerse a lo siguiente:

```
  5 REM TECLEE NUMERO DE PIES
 10 INPUT "PIES?";FT
 15 LET MT=FT*0.3048
 20 PRINT "EN METROS:";MT
 25 LET DM=MT*10
 30 LET CM=DM*10
 35 LET MM=CM*10
 40 PRINT "EN DECIMETROS";DM
 45 PRINT "EN CENTIMETROS";CM
 50 PRINT "EN MILIMETROS";MM
```

Compruebe este programa. Ya sabe las reglas. Cuando encienda su ordenador puede empezar por teclear el programa. Si el ordenador ya está encendido y hay un programa en él, bastará con teclear NEW y pulsar la tecla ENTER, para borrar el programa existente.

Comienze las líneas por un número de línea y teclee las instrucciones. Cuando complete la línea pulse la tecla ENTER.

No se preocupe, por el momento, de las comillas, el punto y coma y los puntos. Todo esto se explicará más adelante. Procure ejecutar este programa y comprobará por qué son tan importante las variables. Observe los nombres de las variables; FT, MT, DM, CM y MM. Estos nombres tienen relación directa con el contenido.

Cada vez que ejecute el programa podrá teclear cualquier dato en pies, por ejemplo: 1,2Ø, 1ØØ, ó 1ØØØ, y el programa le dará las respuestas correctas. Podrá apreciar cómo un buen programa y un ordenador hacen mucho más fácil la tarea que llevaría un tiempo realizar a mano o con una calculadora.

IMPORTANTE: El BASIC del ZX Spectrum requiere que las variables se inicializen y se definan previamente. ¿Qué queremos decir con esto? Mire el siguiente ejemplo:

```
10 LET A=B+1
20 PRINT A
```

Este programa no funcionará porque la variable B no está definida y por ello no la encuentra. El ordenador reaccionará visualizando un mensaje de error: «variable not found» (no encuentra la variable). La mejor forma de arreglar esto es insertando una línea:

```
5 LET B=Ø
```

NOTA: la instrucción INPUT A inicializa la variable A.

NOTA: el mejor lugar de un programa para inicializar las variables es al principio de estos.

Veremos esto más adelante.

## + Y — SON OPERADORES ARITMETICOS

Los símbolos +, —, × y ÷ representan funciones aritméticas familiares para nosotros. Como todo ordenador tiene por misión principal la de manejar números, se hace obvio que estos símbolos son el pan nuestro de cada día.

Por esta razón se denominan Operadores Aritméticos. Después de todo dirigen el flujo del programa hasta cierto punto. Cuando el programador tropieza con el símbolo + normalmente hará la operación de sumar. Lo mismo se aplica para el resto de los símbolos aritméticos. Habrá podido comprobar que el ordenador utiliza distintos símbolos que representan la división y la multiplicación.

En casi todos los lenguajes de programación el asterisco (*) ha sustituido la × del producto. Esto se debe a que el ordenador no distingue entre la × de multiplicar y la letra x. Por lo que se hace necesario otro símbolo para representar la operación.

La barra (/) es el símbolo de división, que no necesita más explicaciones. Es muy común en el mundo de las matemáticas y la aritmética. Aún no hemos tenido en cuenta un operador aritmético, la flecha vertical (↑), cuya función es elevar un número a una determinada potencia. Todos sabemos cómo se eleva un número a una potencia, por ejemplo: para poner 2 elevado a la 3.ª, se escribe: $2^3$. Pero esto es algo que no se puede hacer en el ordenador, por esta razón la flecha se encarga de ello. De manera que si deseamos conocer el resultado de elevar 2 a la 4.ª potencia tendríamos que escribir 2 × 2 × 2 × 2. Para evitar tal engorro, utilizamos la siguiente terminología que cumple a la perfección: 2↑4. Ejecute lo siguiente en su ordenador: PRINT 2↑4 y pulse ENTER.

## EL ORDENADOR TOMANDO DECISIONES

La particularidad que hace que un ordenador sea tan potente en el sentido de que puede hacer muchas cosas, es el hecho de que puede comprobar situaciones y tomar decisiones según los resultados. Estos test (o situaciones) son ejecutadas por los operadores relacionales, en combinación con unos cuantos comandos. ¿Qué es un Operador Relacional? Muy simple, es una función que comprueba la relación existente entre los datos o informaciones. El Operador más conocido es el signo =. Cuando decimos A = B (A igual a B) hacemos una relación entre los elementos A y B. También podemos decir A = B + C. La relación existente entre los dos lados de la igualdad está en que A es igual a la suma de los elementos B y C. También podemos decir que A no es igual a B, o que A es mayor que B, o que A es menor que B.

En cualquier caso probamos la relación existente entre los factores, uno a la izquierda del operador relacional y otro a la derecha.

Que la condición probada por esos operadores relacionales se verifique o no, el ordenador tomará las decisiones necesarias.

Por ejemplo, cuando el ordenador se encuentra con algo similar a:

```
200 LET A=B
210 IF A=10 THEN GO TO 400
220 LET B=B+1
```

Comprobará la relación entre A = 1Ø en la línea 21Ø. Cuando esa relación sea verdadera, esto es, cuando A valga 1Ø, entonces el ordenador dirigirá el flujo del programa hacia la línea 4ØØ. De lo contrario continuará en secuencia.

Hay mucho más en el próximo capítulo. Concluiremos este párrafo nombrando los operadores relacionales que hay además del «=»:

< — singifica MENOR QUE
> — significa MAYOR QUE
< = ó ≤ — significa MENOR O IGUAL
> = ≥ — significa MAYOR O IGUAL
≠ — significa NO ES IGUAL A

Esto junto con las variables es lo más importante en lo que se refiere a la programación porque hacen que su máquina sea lo que es, un ordenador.

## LA FUNCION DE . , : ; y " "

Es posible escribir un programa en BASIC que no disponga de cualquiera de estos símbolos de puntuación. Sin embargo, está comprobado que pueden ser útiles y en algunos casos obligados si se quiere conseguir un programa de buena calidad.

El punto (.) sólo tiene una función. Es la del punto decimal en números que tengan decimales detrás del punto, y nada más. Si lo intenta utilizar al final de las sentencias, el ordenador visualizará una interrogación parpadeante indicándole que no lo entiende.

La coma (,) separa los datos de salida en la sentencias PRINT y en la instrucción DATA (más sobre la instrucción DATA al final de libro).

Pero si por ejemplo, escribe 1Ø INPUT A, B, C, el ordenador le preguntará por tres datos de entrada, uno

detrás del otro (con 16 espacios en blanco entre ellos). Por otro lado en la línea de programa 11Ø PRINT A, B, C, escribira en la pantalla el contenido de A, B y C en la misma línea horizontal, aunque con algunos espacios en blanco. (Pruebe esas dos líneas para ver lo que ocurre).

Los dos puntos (:) separan comandos. Como habrá visto en el capítulo I, las líneas de programas se pueden separar de la forma siguiente:

```
10 INPUT A: INPUT B: INPUT C
```

Tiene el mismo efecto que

```
10 INPUT A
20 INPUT B
30 INPUT C
```

En el primer caso los dos puntos son obligatorios. El ordenador no aceptará

```
10 INPUT A INPUT B INPUT C
```

tampoco

```
10 INPUT A, INPUT B, INPUT C
```

Los dos puntos también se pueden utilizar para crear líneas abiertas de programas, para una mejor legibilidad, como ya se indicó en un punto anterior. Para demostrar esto teclee

```
10 INPUT A
20:
30 LET B=A+C
```

El ordenador ignorará la línea 2Ø durante la ejecución del programa, pero el listado del programa es más legible.

## EL PUNTO Y COMA

El punto y coma (;) puede ser utilizado como separadores en la salida de datos como las comas, pero también se utiliza en combinación con las comillas y el comando INPUT. Por ejemplo:

```
10 INPUT "FECHA DE NACIMIENTO
";DA
```

Compruebe la utilización de las comillas. Cuando ejecute esta línea de programa, el ordenador visualizará por pantalla FECHA DE NACIMIENTO? y un cursor parpadeante a su derecha. Cuando teclee la fecha, se almacenará en la variable DA, gracias al punto y coma. En este caso el punto y coma es obligatorio.

Como separador en una instrucción de salida PRINT, funciona igual que la coma, con la diferencia de que mientras la coma deja un espacio en blanco considerable entre dos resultados, con el punto y coma se puede conseguir que no salgan espacios. Ejecute el siguiente programa:

```
 10 INPUT "DIA DE NACIMIENTO ";
DA
 20 INPUT "MES DE NACIMIENTO ";
MO
 30 INPUT "ANNO DE NACIMIENTO";
YE
 40 PRINT DA;MO;YE
 50 PRINT DA,MO,YE
```

Verá la gran diferencia cuando ejecute este programa.

El punto y coma tienen un efecto muy útil. Teclee el siguiente programa:

```
 5 LET B=0
10 LET A=B+1
20 LET B=A
30 PRINT A;
40 GO TO 10
```

Este es el mismo programa que el aparecido al principio del capítulo I pero con una pequeña diferencia. Ejecútelo y vea lo que ocurre. Se sorprenderá al descubrir lo que hace un punto y coma en la línea 3Ø.

Las comillas ("") son muy útiles cuando quiere imprimir en pantalla palabras, nombres, mensajes, etc. También le ayudarán cuando quiera almacenar una combinación de letras, como nombres en una variable (en ese caso una variable en cadena, pero ya lo veremos más adelante).

Ya habrá intentado algunas aplicaciones. Vea el programa en el párrafo dedicado a las variables. Ahora si quiere listar algo por la pantalla, tendrá que teclear lo siguiente:

```
10 PRINT "Esto es un ejemplo"
```

y pulse ENTER.

El programa que convierte los pies en unidades del sistema métrico utiliza ampliamente los puntos y comas, los dos puntos y las comillas.

Sobre todo, observe que todo carácter situado entre comillas pierde toda su capacidad operativa. Cuando teclee y ejecute

```
10 PRINT "Esto es u ejemplo: A
+ B = C"
```

el ordenador mostrará en la pantalla,

```
Esto es un ejemplo: A + B = C
```

Finalmente, cuando queramos almacenar una combinación de letras, que nosotros denominamos como cadenas (Variables alfanuméricas), tendrá que ponerlos entre comillas de nuevo y almacenarlas en variables cuyo nombre contenga el signo $.

Por ejemplo, si quiere almacenar los nombres de las ciudades, el programa será algo parecido a lo siguiente:

```
10 LET A$="AMSTERDAM"
20 LET B$="BERLIN"
30 LET C$="COLONIA"
40 LET D$="DOVER"
50 PRINT A$;B$;C$;D$
```

Para finalizar el capítulo veamos la instrucción REM.

## INSTRUCCION REM

Ya se adelantó que REM significa comentario (del inglés REMark). El único próposito es hacer comentarios sobre los programas que está a punto de escribir. No es un comando pero dentro de la estructura de un programa es muy importante, porque permite seguir con precisión lo que está a punto de hacer y el objeto de dicho paso en el programa.

Se dice que la sentencia REM se puede colocar en cualquier sitio dentro de un programa. Esto no es totalmente cierto. por ejemplo, las siguientes líneas de programa no ejecutarán.

```
10 LET A=2: REM Esta linea no
funcionara: LET B=4*A: PRINT B
```

Al ejecutar esta línea el ordenador le contestará con un «ready» sólo porque no ha ido más allá de LET A = 2. Cuando el ordenador se encuentre con la sentencia REM, ingnorará todo lo que le sigua. Lo siguiente sí funcionará:

```
 10 LET A=2: REM Esto si funcio
nara
 20 LET B=4*A: PRINT B
```

La instrucción REM puede colocarse al final de una línea de programa, o se puede utilizar como una línea separada:

```
10 LET A=2
15 REM Esto funcionara igual
20 LET B=4*A: PRINT B
```

## EJERCICIOS

Ya está usted listo para escribir su primer programa, con todo lo que ha visto hasta ahora podrá hacerlo. Cerraremos este capítulo con un ejemplo práctico y al final hará un pequeño problema por su cuenta.

```
 10 REM ESTE PROGRAMA CONVIERTE
 LIBRAS EN CUALQUIER UNIDAD DEL
 20 REM SISTEMA INGLES DE PESOS
 30 INPUT "TECLEE NUMERO DE LIB
RAS ";LB
 40:
 50 LET SE=LB/14: REM STONES =
LIBRAS / 14
 60 LET ON=LB*16: REM ONZAS = L
IBRAS * 16
 70 LET GR=ON*16: REM GRAINS =
ONZAS * 16
 80:
 90 PRINT : REM LINEA EN BLANCO
100 PRINT "EN STONES:";SE
110 PRINT "EN ONZAS:";ON
120 PRINT "EN GRAINS:";GR
130 STOP
```

## RESOLVER

Sabiendo que 1 lb. = 453.6 gramos, haga un programa que convierta Stones, libras, onzas y granos en gramos. Oriéntese con el ejemplo anterior. Para una posible solución ver el Apéndice.

## CAPITULO 3:

# El organigrama es una buena ayuda

Este capítulo le introducirá a un concepto muy útil en lo que se refiere a la programación, el organigrama. Al mismo tiempo continuaremos explicando nuevas herramientas del BASIC. Verá las instrucciones del tipo IF-THEN porque son las más fáciles de entender con un organigrama. Muy relacionados con la combinación IF-THEN están los comados GO TO y GO SUB, que explicaremos detalladamente más adelante.

## EL CONCEPTO DE ORGANIGRAMA

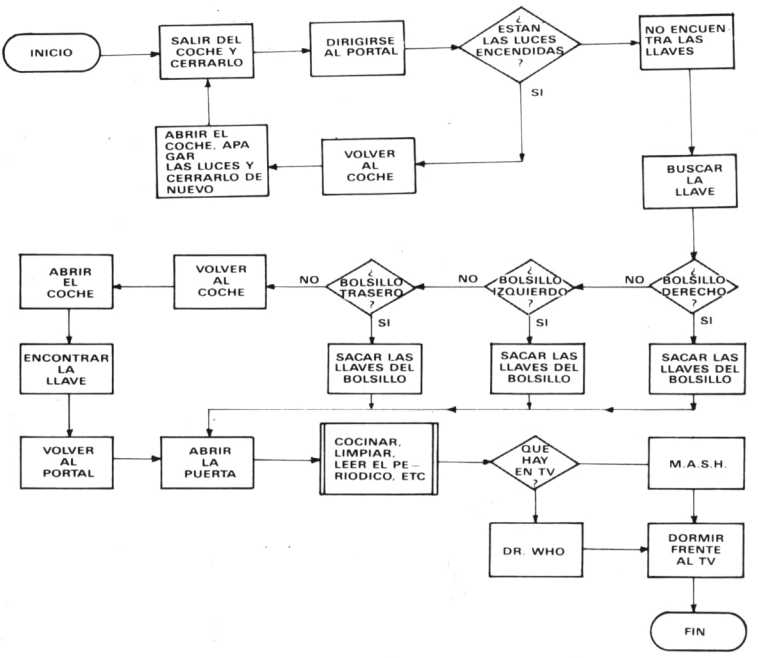
¿Se ha dado cuenta de que su vida diaria es un continuo tomar decisiones y actuar según éstas? La mayoría de nosotros no somos conscientes de ello pero desde el mismo momento en que nos levantamos por la mañana, desayunamos, vamos al colegio o a trabajar, etc., seguimos una trayectoria más o menos fija. En otras palabras, desarrollamos un programa.

Utilizamos las acciones que hacemos durante el día para explicar el concepto de organigrama. Tomaremos como ejemplo una determinada hora del Sr. López. Primero lo describiremos con palabras, detallando cuidadosamente las secuencias de las acciones y decisiones.

El Sr. López es un solterón divertido que está volviendo a casa. Para el coche, sale de él y cierra la puerta con llave. Pero a mitad de camino se pone a pensar si habrá dejado las luces del coche encendidas. Se da la vuelta y efectivamente, las luces del coche están encendidas. Vuelve al vehículo, abre la puerta y apaga las luces. Cierra nuevamente el coche y se dirige a casa, pero al llegar a la puerta, de repente se da cuenta de un nuevo problema: ¿Dónde está la llave? Se habrá caido del llavero. Prueba buscar en todos sus bolsillos pero recuer-

da que la llave de casa se le cayó cuando estaba conduciendo. Pudo haber ocurrido otra vez. Vuelve al coche, lo abre y encuentra la llave en el suelo. Más tranquilo, cierra el coche y se dirige a casa entrando esta vez sin ningún problema. Una vez dentro, decide comer una pizza. La saca del congelador y la mete en el horno. Después de cenar y de fregar los platos (le gusta el orden) decide ver un programa de TV. ¿Qué verá?; ¿M.A.S.H. o el Dr. Who? M.A.S.H. le parece más divertido, pero se duerme frente al televisor.

Ahora vuelva a leer la historia pero esta vez usando el diagrama de flujo, ponga mucha atención en la relación existente entre las acciones escritas entre los rectángulos y las decisiones que son los rombos. Siga el sentido de la flecha.




Cuando lo haya hecho discutiremos más detenidamente los símbolos del diagrama.

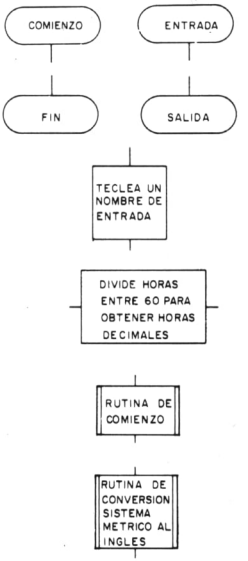
## EXPLICACION DE LOS SIMBOLOS DEL DIAGRAMA

En el diagrama de flujo de la vida del Sr. López, se han utilizado tres símbolos:

Este símbolo, ⬭ denominado terminal, siempre va situado al comienzo y al final del diagrama. Cuando se usa al principio, podemos escribir «comienzo» o «start», y cuando se utiliza al principio de una subrutina podrá escribir «entrada». Al final del diagrama podrá escribir «fin» o «stop». Y al final de una subrutina se escribirá «retorno».

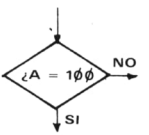
El rectángulo significa acción a tomar, o como se dice en la jerga de los ordenadores, un proceso. La frase «Sale del coche y lo cierra» no es nada más que la descripción de un acción. El objeto del diagrama del flujo nos parece suficientemente claro: describimos un proceso dentro del rectángulo del diagrama y más adelante se traducirá a un lenguaje de programación que entienda el ordenador.

Este rectángulo ▭ es una variación del anterior. Indica un proceso predefinido que significa nada más que lo que contiene dentro es otro programa entero, esto es, una subrutina. Cada vez que se encuentre con un símbolo semejante en un diagrama, significa que en otra parte hay un organigrama describiendo el proceso a seguir.

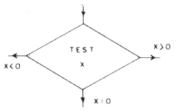


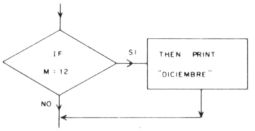

Compruebe de nuevo el organigrama del Sr. López. Hay un símbolo que significa «cocinar, limpiar, leer el periódico, etc.». Un juego completo de acciones dentro de un bloque requiere un organigrama separado, o mejor dicho una subrutina. En el capítulo siguiente aprenderá cómo utilizar este símbolo.

El rombo es muy importante porque hace referencia a la verdadera potencia dentro de un ordenador, esta es la habilidad de poder tomar decisiones. Cada vez que se encuentre con este símbolo, tendrá que tomar una decisión. Significa que el programador tendrá que desarrollar sus programas de forma que el ordenador pueda tomar decisiones. Dependiendo de esta, el programa irá en un sentido o en otro, indicado por el «si» y «no». Este símbolo lo verá en las partes del programa donde hay que tomar decisiones.




Volvamos al organigrama del Sr. López. La primera decisión con la que tropieza es la de si las luces del coche están encendidas o no. Si es cierto, la condición anterior se verifica. La respuesta es «SI» y decimos que la



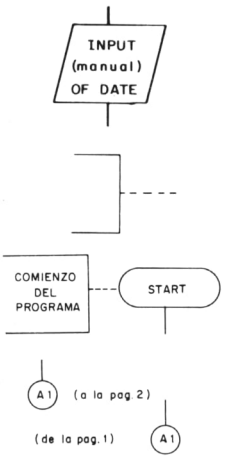




condición es verdadera. En ese caso, el flujo del programa vuelve al principio del proceso. O, como decimos, el programa se bifurca.

Bien, veamos al Sr. López. La condición es verdadera, por lo que vuelve al coche y apaga las luces. Cuando se dirigía a su casa podría haberse preguntado de nuevo si las luces del coche seguían encendidas. Entonces la respuesta sería «NO». La condición ahora sería falsa. Como consecuencia de ello el programa no se bifurca sino que sigue en secuencia.

Observe que la palabra IF está en mayúsculas. Esto es así para resaltar que en situaciones del tipo IF-THEN es donde esta combinación resulta más útil. Veremos más.

## MAS SIMBOLOS DEL ORGANIGRAMA




El paralelogramo significa entrada (INPUT) o salida (OUTPUT). De esta forma puede indicar en su programa cualquier tipo de entrada de datos, mediante el comando INPUT, o alguna forma de salida de datos, utilizando la pantalla o la impresora. Como este símbolo puede significar tanto entrada como salida de datos, es aconsejable que indique el tipo de proceso que va a hacer, ya sea de entrada o de salida de datos.

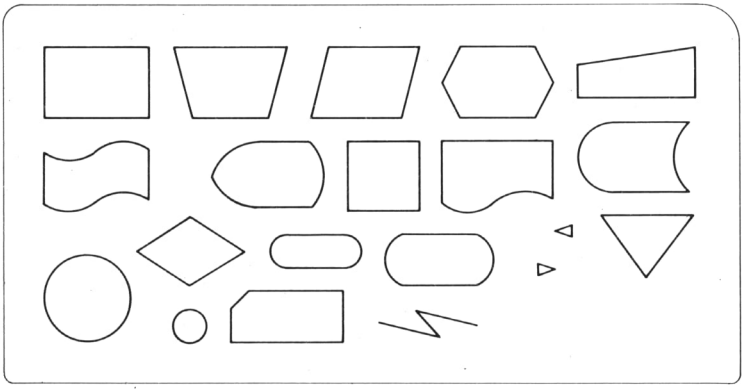
Símbolo de anotaciones ⊐---. Muchas veces no tenemos espacio suficiente dentro de un símbolo para especificar el tipo de procedimiento que vamos a hacer. En ese caso utilizamos este símbolo para añadir comentarios extras.

Este símbolo es O el conector. Algunas veces los organigramas se complican (procure evitarlo siempre). Una forma de que su organigrama no se convierta en una tela de araña es mediante el uso de los conectores en lugar de unas líneas largas de flujo. Entonces se coloca un conector desde el lugar en que estamos al lugar donde queremos ir. Ambos conectores tendrán el mismo código dentro de ellos para indicar el correcto flujo del programa.

Hay muchos símbolos más dentro de un organigrama pero con los que ya ha visto puede hacer la mayoría de los trabajos.

Naturalmente, los organigramas han de hacerse en papel y a mano. Pero normalmente esto acaba siendo un trabajo engorroso, para lo cual le sugerimos la utilización de una plantilla con los símbolos del organigrama,

como el que puede ver a continuación. Lo más seguro es que el establecimiento donde compró el ordenador los venda; si no, diríjase a alguna casa especializada en material de dibujo o a una tienda de ordenadores. El que mostramos en esta página es de la firma alemana FABER CASTELL. Son muy manejables y dan a su organigrama un aire profesional.



## PROS Y CONTRAS DE LOS ORGANIGRAMAS

Como en todas las cosas siempre hay dos caras en la moneda. Esto también se aplica a los organigramas —tiene una gran ventaja pero también sus desventajas.

Dejemos claro que la importancia del organigrama depende del lenguaje de programación que vamos a utilizar con él. Al contrario que con el BASIC, hay muchos lenguajes de programación que no son interactivos. En estos casos los organigramas tienen que ser muy elaborados y es obligatorio antes de que el programador vaya más adelate con el ordenador. Los lenguajes que no son interactivos no dejan mucha opción en lo que se refiere a las situaciones de condición. El programador, sin embargo, tiene que hacer el papel del ordenador cuando hace el organigrama, para probar el correcto funcionamiento del programa antes de codificar el organigrama en el lenguaje correspondiente.

Con el BASIC, por otro lado, se pueden hacer tests contínuamente acerca de lo que se ha escrito, por lo que

muchos de los programadores del BASIC no necesitan el organigrama. La mayoría no se equivoca, pero hay una minoría que opina lo contrario. El organigrama no se puede olvidar completamente porque tiene múltiples ventajas, entre las cuales podemos citar las siguientes:

- la principal ventaja de un organigrama es la visualización del programa que vamos a codificar.
- puede ayudar a encontrar soluciones a problemas complicados. Por ejemplo, en el análisis de partes enrevesadas de un programa, especialmente aquellas donde se toman muchas decisiones (esto estará más claro en cuanto expliquemos la combinación IF-THEN).
- un organigrama general del problema nos dará una idea de cómo establecer las rutinas principales y las subrutinas.

Es aconsejable que trabaje la base del organigrama general cuando esté diseñando un programa. Esto entra dentro de la idea de programación estructurada.

¿Y sobre el organigrama detallado? Es en esta parte del libro donde nos desviamos de la opinión general, porque es aquí donde con más claridad podremos comprobar su desventaja.

El organigrama es muy útil a la hora de confeccionar programas pequeños, pequeñas subrutinas y probar flujos complicados de programas. Un organigrama del tamaño de un folio (tipo dyn a4) puede establecer una visión clara del problema, con la condición de que se tiene que dibujar claramente.

Llegados a este punto conviene aclarar que muchos programadores no tienen el talento suficiente a la hora de hacer organigramas claros y fáciles de seguir. Se complica todavía más cuando el programa en cuestión es largo y pesado y el organigrama ocupa varias hojas. No es nada agradable leer un organigrama que ocupa unos veinte o más folios. En pocas palabras, un organigrama demasiado largo se convierte en un estorbo, por lo que los programadores prefieren ignorarlos, aunque sea de un programa que haya que actualizar. De manera que un organigrama estructurado y bien escrito, que incluya comentarios, es más fácil de seguir que uno largo.

Obviamente, el dibujar un organigrama largo se convierte en una tarea tediosa y una pérdida de tiempo, concretamente cuando uno lo quiere hacer lo mejor posible. Cuando un programa se actualiza, igualmente habrá que actualizar el organigrama. Esto implica rehacer el dibujo y es natural que uno se olvide esto. Por lo que

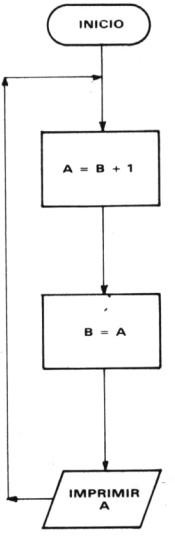
después de un tiempo el organigrama no representará la realidad del programa.

Por otro lado, cuando un programador actualiza un programa es más fácil introducir líneas de comentarios. Un buen diseño de programa, esto es programación estructurada, siguiendo una realización lineal, se puede seguir sin un organigrama extenso. Recomendamos que el organigrama se haga de forma limitada, que sólo es una herramienta para ayudarnos a conseguir mejores programas. En su forma limitada, como se explicó en párrafos anteriores, puede ser muy útil; en su forma detallada y extendida se convierte en una carga innecesaria. En general consideramos al organigrama como opcional.

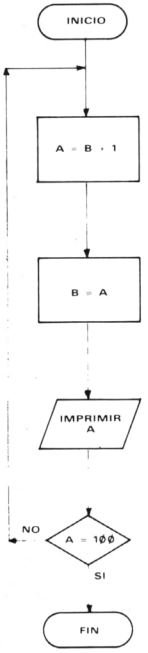
## IF-THEN

¿Se acuerda de lo que dijimos acerca de los bucles sin fin? (Capítulo 1). Aquí tenemos de nuevo aquel programa que sólo contaba:

```
10 LET B=0
20 LET A=B+1
30 LET B=A
40 PRINT A
50 GO TO 20
```




La idea del bucle está clara en el organigrama, pero es menos claro en las cinco líneas del programa. En esencia es un bucle sin fin. Si no fuera por la pregunta que le hace su ZX - Spectrum «scroll», que para el programa en cuanto la pantalla está llena con los resultados del ciclo ejecutado hasta ese momento, el programa ejecutaría sin parar. En estos casos sólo se puede parar desconectando el ordenador, o pulsando la tecla BREAK. Pero no es una solución muy elegante. Es mejor poner una condición de interrupción en el programa. Hacemos esto con la técnica ya explicada que es la de «bifurcaciones condicionales». En el programa se comprueba una condición y si el resultado es cierto, entonces el programa se bifurca y para. Ahora verá el uso de la condición IF-THEN.




```
10 LET B=0
20 LET A=B+1
30 LET B=A
40 PRINT A
50 IF A=100 THEN GO TO 70
60 GO TO 20
70 STOP
```

Lo que ocurre en este segundo ejemplo es que en cada bucle el número almacenado en la variable A es comparado en la línea 5Ø. El test se limita a comparar el número almacenado en A con el número 1ØØ. Si la condición es cierta (la respuesta sea SI) THEN (entonces) el ordenador dirigirá el flujo del programa hacia la línea 7Ø. O como dice la línea THEN GO TO 7Ø. El programa bifurca en este punto. Pero mientras la condición no se cumpla (la condición sea falsa) el programa ejecutará la línea siguiente (5Ø). Para entender claramente los pasos seguidos sugerimos que teclee el programa y lo ejecute (RUN). Incluso puede alterar la línea 5Ø unas cuantas veces, por ejemplo IF A = 25 THEN GO TO 7Ø, o IF A = 1ØØØ THEN GO TO 7Ø, etc.

La combinación IF-THEN, se lee de la manera siguiente: IF (condición), THEN (acción a tomar en caso de que la condición sea verdadera).

Entre IF y THEN puede colocar cualquier operador relacional (=, and, etc.) mientras que detrás de THEN sólo se puede colocar una palabra clave (el BASIC del ZX Spectrum es muy estricto en este sentido, otros ordenadores son más flexibles en estas sentencias). Observe de nuevo la línea 5Ø del programa anterior:

5Ø IF A = 1ØØ THEN GO TO 7Ø

GO TO es una palabra clave, como recordará.

En cuanto a los resultados del programa, obtendrá los mismos si cambia la línea 5Ø por:

5Ø IF A = 1ØØ THEN STOP

Cuando la condición A = 1ØØ sea verdadera el programa finalizará en la línea 5Ø en vez de en la 7Ø.

Aunque parezca fácil, esta forma de acabar los programas no es nada recomendable, porque no entra en los esquemas de la programación estructurada.

El comando STOP, en medio de la nada, por decirlo de alguna forma, hará el trabajo de parar el programa pero no lo hará más legible. En este aspecto tenemos que añadir que el ZX Spectrum no usa el comando END tan utilizado en otros ordenadores. El comando END fi-

naliza totalmente el programa mientras que STOP sólo lo interrumpe. A continuación tiene dos líneas de programación con los usos válidos de la combinación IF-THEN.

```
100 IF A=10 THEN PRINT "OCTUBRE
"
100 IF A=10 AND B=20 THEN GO SU
B 1000
```

NOTA: la última línea usa el operador AND y el comando GO SUB. Explicaremos más sobre el operador AND en capítulos sucesivos.

NOTA: se preguntará por qué no se ha inicializado en el organigrama la variable B. Esto se debe a que el proceso de inicialización de variables se considera como una operación estándar, por lo que no se indica.

## EL COMANDO GO TO

Se habrá cruzado varias veces con el comando GO TO, por lo que no tendrá muchos problemas a la hora de entender su significado. Su funcionamiento es obvio. Dirige el flujo del programa hacia otra parte del mismo programa, indicado por el número de línea a continuación del comando. Cuando se utiliza en combinación con el binomio IF-THEN, nos referimos a bifurcaciones condicionales, porque depende de la condición a probar el que el programa continúe en secuencia o se dirija a otra parte. Pero si se utiliza el comando GO TO a secas con un número de línea a continuación, tenemos lo que se denomina una bifurcación incondicional. Echele un vistazo al siguiente programa:

```
10 INPUT A
20 INPUT B
30 LET C=A+B
40 GO TO 100
50 LET D=A-B
```

En la línea 4Ø el programa se encuentra con el comando GO TO 1ØØ y como consencuencia de ello bifurca a la instrucción 1ØØ, sin haber hecho ningún test. En casos como este las líneas entre la 4Ø y la 1ØØ

no ejecutan nada a no ser que en alguna parte después de la línea 1ØØ nos dirijamos a la instrucción inmediatamente después de la 4Ø.

En cualquier caso el abuso del comando GO TO es una señal clara de programación pobre. Hay otras técnicas de bifurcación más elegantes que se verán detenidamente.

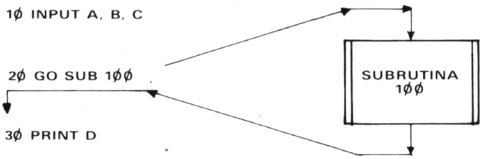
## LA COMBINACION GO SUB-RETURN

Cuando conozca todas las posibilidades del comando GO TO, el siguiente paso será el estudio de la combinación de GO SUB-RETURN. GO SUB significa GO (ir) a una SUB (rutina) y tiene que estar seguido de un número de línea. La función del RETURN se verá más adelante. Para entender mejor el significado del comando GO SUB lo compararemos con una experiencia diaria. Mientras que GO TO significa «ir a un lugar determinado y quedarse allí hasta que se reciba otra orden que indique lo contrario», GO SUB significa «ir a un lugar determinado, ejecutar un trabajo asignado, volver (RETURN) inmediatamente y esperar nuevas órdenes».

Es como hacer un trabajo de carpintería; de repente nuestro carpintero se da cuenta de que no le quedan más clavos. Ha empezado con 1ØØ clavos y cuando los pone todos se da cuenta de que no tiene los suficientes como para acabar el trabajo. Su programa le indica que tiene que dirigirse a una ferretería, comprar otros 1ØØ clavos y volver (RETURN) para continuar con el trabajo. Ahora comprobará que el programa es poner los clavos, mientras que el ir a la ferretería es una subrutina. Naturalmente esta es una analogía que no aclara lo suficiente lo que es una subrutina —lo único que demuestra es la inexperiencia del carpintero, mientras que el objetivo de una subrutina es optimizar la programación. Pero esta analogía enseña la forma en que trabaja— ir a un sitio determinado y volver inmediatamente. En un programa trabajaría de la siguiente manera:

```
  10 INPUT A,B,C: REM INPUT DE H
R, MN Y SEG
  20 GO SUB 100: REM CONVERTIR A
 HORAS DECIMALES
  30 PRINT D    : REM IMPRIMIR CO
NVERSION
```

Lo que ocurre en este programa es que en la línea 2Ø se dirige a una subrutina donde aparentemente las variables A, B y C, que contienen las horas, minutos y segundos, son convertidas a horas decimales y almacenadas en la variable D. Cuando esta operación finaliza, el programa vuelve a la línea 2Ø pero ejecuta la línea inmediatamente, la 3Ø. el diagrama lo aclarará:




Recuerde esto: el comando GO SUB sólo trabaja correctamente cuando al finalizar las operaciones se encuentra con un RETURN.

En nuestro ejemplo, la subrutina 1ØØ sería algo parecido a esto:

1ØØ REM ESTA SUBRUTINA CONVIERTE...

15Ø RETURN

IMPORTANTE: una subrutina debe finalizar con el comando RETURN. De lo contrario el programa haría cosas raras, porque el puntero del programa no encuentra el camino de retorno, por lo que empieza a irse por otras partes del programa.

Como ya se dijo, las subrutinas están para optimizar la programación. Para ver lo que queremos decir, estudie el siguiente problema.

Si alguna vez ha hecho operaciones aritméticas con horas, minutos y segundos, entonces sabrá lo confuso que es trabajar no con decimales sino con hexadecimales, esto es, la división de un número en unidades de sesenta en vez de diez. El problema que esto supone se puede resolver convirtiendo horas, minutos y segundos en horas decimales antes de hacer cualquier operación aritmética. Es obvio que esta tarea la puede hacer su ordenador.

La manera de convertir un número dado de horas, minutos y segundos a decimales es: primero divida los segundos entre 6Ø, sume el resultado a los minutos, divi-

da de nuevo los minutos entre 6Ø, y sume el resto a las horas, que se convertirán en decimales. Una rutina que resuelva esto tendrá el siguiente listado:

```
290 REM DIVIDE SEGUNDOS / 60
300 LET SE=SE/60
310 REM SUMA SEG. DEC. A MIN.
320 LET MN=MN+SE
330 REM DIVIDE MIN. / 60
340 LET MN=MN/60
350 REM SUMA MIN. A SEG.
360 LET HR=HR+MN
```

Ahora hagamos un programa que sume tres grupos de horas, minutos y segundos.

```
 10 INPUT "HORAS ";A
 20 INPUT "MINUTOS";B
 30 INPUT "SEGUNDOS";C
 40 LET C=C/60
 50 LET B=B+C
 60 LET B=B/60
 70 LET A=A+B
 80 INPUT "HORAS";D
 90 INPUT "MINUTOS";E
100 INPUT "SEGUNDOS";F
110 LET F=F/60
120 LET E=E+F
130 LET E=E/60
140 LET D=D+E
150 INPUT "HORAS";G
160 INPUT "MINUTOS";H
170 INPUT "SEGUNDOS";I
180 LET I=I/60
190 LET H=H+I
200 LET H=H/60
210 LET G=G+H
220 LET K=A+D+G
230 PRINT K
240 REM SALIDA DE HORAS DEC.
```

Como puede ver, el programa tiene tres juegos completos de entradas (INPUT), que son las líneas 1Ø, 2Ø y 3Ø, líneas 8Ø, 9Ø y 1ØØ y líneas 15Ø, 16Ø y 17Ø respectivamente. Tenga en cuenta las líneas de la 4Ø a la 7Ø, de la 11Ø a la 14Ø y de la 18Ø a la 22Ø. Estos tres grupos de líneas hacen la misma tarea. El programa funcionará bien pero es muy ineficaz.

Es aquí donde una subrutina, la combinación de GO SUB y RETURN, nos puede salvar la situación.

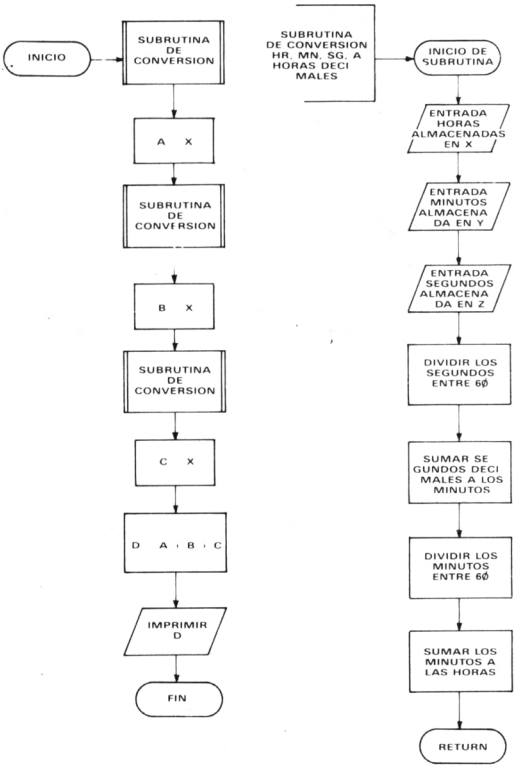
Teclee el siguiente programa y ejecútelo:

```
  10 GO SUB 100
  20 LET A=X
  30 GO SUB 100
  40 LET B=X
  50 GO SUB 100
  60 LET C=X
  70 LET D=A+B+C
  80 PRINT D
  90 GO TO 1000
 100 INPUT "HORAS";X
 110 INPUT "MINUTOS";Y
 120 INPUT "SEGUNDOS";Z
 130 LET Z=Z/60
 140 LET Y=Y+Z
 150 LET Y=Y/60
 160 LET X=X+Y
 170 RETURN
1000 STOP : REM FIN DEL PROGRAMA
```

En la columna de la izquierda está la parte principal del programa (líneas 1Ø a 9Ø) y a la derecha están las subrutinas (líneas 1ØØ a 17Ø). Es esa subrutina la que hace el trabajo. Siga el orden de las líneas.

En la línea 1Ø se encuentra el comando GO SUB 1ØØ. Por lo que el puntero del programa se dirige a la línea 1ØØ, donde ejecuta la subrutina. En esta subrutina las horas, minutos y segundos son convertidos a horas decimales, y en la línea 17Ø el programa se encuentra con el comando RETURN. El control vuelve al programa principal a la línea 2Ø donde guarda el resultado de la subrutina en la variable A. En la línea 3Ø el programa salta de nuevo a la subrutina y vuelve a la línea 4Ø donde almacena el resultado de las operaciones en la variable B, y así sucesivamente. Este programa es más eficiente que el anterior que hace el mismo trabajo. Por eso el concepto de subrutina es tan importante en la programación de ordenadores. Ningún problema medianamente grande estará sin ellas.

Finalizamos el capítulo con un organigrama del último programa. Estúdielo con sumo cuidado, porque en el capítulo siguiente comenzará el trabajo real.




- Comienzo del programa
  GO TO a la subrutina de hr-mn-sg/horas decimales
  RETURN al programa principal.
- Almacena el resultado de la conversión en la variable A.
  GO TO a la subrutina de (hr-mn-sg/horas decimales.
  RETURN al programa principal.
- Almacena el resultado de la conversión (hr-mn-sg/hr. dec.) en la variable B.
  GO TO a la subrutina de conversión hr-mn-sg/horas decimales.
  RETURN al programa principal.
- Almacena el resultado de la conversión (hr-mi-sg/hr.dec.) la variable C.
- Suma A a B + C
- Almacena el resultado en la variable D
- Imprime el contenido de la variable D
- Fin de programa

**Subrutina de conversión de hrs., min., seg. a horas deciamales**

- Entrada del número de horas, almacenado en la variable X
- Entrada del número de minutos, almacenado en la variable Y
- Entrada del número de segundos, almacenado en la variable Z
- Dividir los segundos entre 6Ø
- Sumar los segundos decimales a los minutos
- Dividir los minutos entre 6Ø
- Sumar los minutos a las horas
- RETURN al programa principal

## CAPITULO 4:

# Preparando los programas

Cuando haya estudiado detenidamente los capítulos anteriores habrá adquirido los conocimientos suficientes como para empezar a diseñar y escribir sus propios programas por muy simples que sean. En este capítulo aplicará lo que ha aprendido hasta el momento.

Describiremos detalladamente la estructura lineal dentro de la programación estructurada.

Todo se reduce a lo siguiente: Piense antes de proseguir. Cuanto antes empieze a escribir sus programas, en vez de planearlos detenidamente, más tiempo invertirá en hacer un buen programa. Un programa bien diseñado y escrito es en un 7Ø% el resultado de pensar y en un 3Ø% de codificación (escribirlo).

### PRIMERA TAREA: CONVERSION DE PESOS Y MEDIDAS

Este capítulo y el siguiente estarán dedicados al diseño de programas que convierten los pesos y medidas del sistema inglés al sistema métrico y viceversa. Hemos escogido este problema para aplicarlo al ordenador, no sólo por su interés general sino también porque los cálculos y procedimientos son simples, por lo que la programación no será complicada. Es más, con los conocimientos que tenemos de los capítulos anteriores estamos en condiciones de escribir un buen programa.

### RESUMEN DE LOS PASOS A SEGUIR

Vamos a resumir los procedimientos que tendrá que seguir de ahora en adelante (los detalles se explicarán más adelante):

**(1)** Definir y especificar el objeto del trabajo —esto incluye la documentación.

**(2)** Diseñar el formato de lo que crea será la salida del programa.

(3) Diseñar el formato de lo que va a ser la entrada de datos.

(4) Trabajar con bloques individuales (o rutinas o módulos que ya explicaremos lo que son) que crea necesarios para completar el trabajo.

(5) Desarrollar la parte principal del programa —preferentemente con un organigrama. Desarrollar los siguientes niveles, también con organigrama.

NOTA: Como ya se explicó, el organigrama es opcional. Si decide trabajar sin él, entonces escriba las secuencias de los procedimientos que crea necesarios.

Hasta aquí lo que se refiere al diseño del programa, ahora veamos la codificación:

(6) Es esencial dentro de la estructura de la programación lineal empezar desde arriba, según describimos en el organigrama, y en consecuencia trabajar hacia abajo.

(7) La parte principal será la más importante —no debe contener rutinas de programación. Las rutinas se deben situar en la zona dedicada a ellos.

(8) Proceder a desarrollar el nivel siguiente sólo cuando haya comprobado y verificado el correcto funcionamiento de la rutina que esté desarrollando; asegúrese de que funciona.

(9) Dejar los refinamientos para el final. Déle prioridad a la parte de programación, que funcionen todas y cada una de las partes y en consecuencia el programa entero.

(1Ø) Cuando el ordenador se lo permita, esto es, cuando tengan memoria suficiente, procure insertar la instrucción REM para aclarar pasos y mejorar la legibilidad.

(11) Aunque no sea una parte esencial de la programación, siempre es bueno inicializar el programa con una pequeña sección de identificación. Para mejorar la documentación del programa, es importante que todas las variables utilizables sean descritas detalladamente.

(12) Para aquellos que tengan impresoras: escriba sus programas con un formato claro. Cuando imprima el programa verá la diferencia.

## PRIMER PASO: DEFINIR DETALLADAMENTE EL TRABAJO

Considérese como su propio cliente. Es el programador y se pone la tarea de escribir un programa que con-

vierta los pesos y medidas del sistema inglés al sistema métrico y viceversa. Es fundamental que el programa no sólo lo entienda usted sino también los clientes. ¡No se asuste! No piense que no es capaz de hacerlo. Lo conseguirá porque es más fácil de lo que usted cree. Por otro lado si es optimista y piensa que acabará en una hora, entonces está completamente equivocado. Cualquier trabajo de programación siempre es más largo de lo que uno cree, particularmente cuando no se emplea tiempo averiguando lo que se le pide hacer. De manera que antes de tocar el ordenador defina el trabajo detalladamente, hasta el más mínimo detalle. No es suficiente decir que convierta pesos y medidas del sistema inglés al métrico. Esto no se puede considerar como una definición del trabajo. Tendrá un grave problema cuando compruebe que no entiende lo que quiere el cliente (aunque el cliente sea usted, como en este caso). Puede estar acabando el programa y de repente descubrir que se ha olvidado..., o que es mejor dejar..., etcétera.

Definamos con detalle el presente problema.

Cuando los requerimientos son la conversión de los pesos y medidas del sistema inglés al métrico y viceversa, la primera pregunta es ¿qué pesos y qué medidas? Por lo que deberá aclarar el tipo de pesos y medidas a convertir.

La pregunta siguiente será ¿cómo lo queremos convertir? Esto es, si se desea la opción para, por ejemplo, teclear cada valor estándar del peso y medida y obtener en la salida todas las conversaciones del dato. ¿O quiere teclear el peso o medida correspondiente? Vamos a aclarar esto. En la primera opción podemos pedir al programa que resuelva el problema ¿cuántos centímetros hay en 15 pulgadas? o ¿cuántas yardas hay en 1ØØ metros? o ¿cuántas toneladas hay en 1.ØØØ kilogramos? etc. Claro que un programa que envíe esa información es viable pero dese cuenta que es más complicado que la otra opción: entrada de la medida y/o peso estándar, produciendo así todas las conversiones en el sistema métrico e inglés. Escogeremos la segunda opción. Para la longitud, tendremos metros y pies como datos de entrada y como datos de salida tendremos todas las conversiones posibles; en cuanto al peso, tendremos kilogramos y libras como datos de entrada y como datos de salida todas las conversiones posibles en ambos sistemas.

Esta especificación no es suficiente. Hay que definir qué tipo de medida vamos a necesitar.

Aqui nos encontramos con otra parte importante que necesita un programa, la documentación. Un problema no sólo ha de estar propiamente definido, sino que también ha de estar bien documentado. Un caso como el que nos ocupa aclara lo que queremos decir. Si quiere convertir adecuadamente las medidas via ordenador, necesitará saber la fórmula exacta de la conversión. Tendrá que ojear libros y averiguar a cuantos centímetros equivale una pulgada, o cuantas libras hay en un kilogramo, etc. Por esa razón adjuntamos un cuadro debidamente cumplimentado:

TABLA DE PESOS Y MEDIDAS (inglés y métrico)

| | | | | |
|---|---|---|---|---|
| (PESOS) | | | | |
| 1 tonelada inglesa (T) | = | 20 cwt (hundredweight) | = | 1016 kg. |
| 1 hundredweight (cwt) | = | 4 qr (quarters) | = | 50.8 kg. |
| 1 quarter (qr) | = | 2 st (stones) | = | 12.7 kg. |
| 1 stone (st) | = | 14 lb (libras) | = | 6.38 kg. |
| 1 libra (lb) | = | 16 oz (onzas) | = | 453.6 gramos |
| 1 onza (oz) | | | = | 28.35 gramos |
| 1 kilogramo | = | 1000 gramos | | |
| 1 hectogramo | = | 100 gramos | | |
| 1 decagramo | = | 10 gramos | | |
| (MEDIDAS DE LONGITUD) | | | | |
| 1 milla naútica (nudo) | = | 2026 yardas | = | 1852 metros |
| 1 milla | = | 1760 yardas | = | 1609.315 metros |
| 1 yarda | = | 3 pies | = | 0.9144 metros |
| 1 pie | = | 12 pulgadas | = | 0.3048 metros |
| 1 pulgada | = | | | 0.0254 metros |
| 1 kilómetro | = | 1000 metros | | |
| 1 hectómetro | = | 100 metros | | |
| 1 decámetro | = | 10 metros | | |
| 1 metro | = | 100 centímetros | = | 1000 milímetros |
| 1 decímetro | = | 10 centímetros | = | 100 milímetros |
| 1 centímetro | = | 10 milímetros | | |

Con esta tabla podrá convertir cualquier cantidad que tenga pensada. Es más, podrá decidir cual será el modelo estándar que utilizará a lo largo del programa. Ya lo hemos decidido por usted: todos están incluidos en programa.

Resumamos las especificaciones del trabajo:

- Conversión de pesos y medidas del sistema inglés al sistema métrico y viceversa.
- Entrada de datos estándar: pies y metros para la longitud; kilogramos y libras para el peso.
- Salida de datos en los dos sistemas, para cada opción (longitud y peso).

Una última aclaración antes de seguir adelante. Está claro que los programas los escribe para disfrute propio. Pero evite que el programa sólo lo pueda leer usted. Un buen programa es aquel que, entre otras cosas, puede ser entendido por otros cuando se ejecuta. Nunca suponga que otros entenderán lo que quiso decir. Nunca piense «esto es obvio». Lo que está claro para usted no está claro para otros. Por esta razón muchos programas buenos son un desastre comercial porque no son legibles por el usuario. Esto significa que un programa tiene que ser facilmente accesible a cualquier usuario y no sólo al programador que lo escribió.

## SEGUNDO PASO: DISEÑAR EL FORMATO DE SALIDA

La razón de porqué hemos puesto este párrafo a continuación de la definición del trabajo es porque uno es parte del otro. El objeto principal del programa es lo que tiene que devolver al usuario, un resultado, esto es, OUTPUT (salida). Como ya sabe la salida de datos se puede presentar de dos formas, en papel o por pantalla. En cualquiera de los casos, la salida tiene que ser clara. Tiene que estar claro lo que significa una figura determinada escrita en pantalla o en papel. Si por ejemplo usted escribe un programa como este:

```
10 INPUT A
20 LET G=A*453.6
30 PRINT G
```

y lo ejecuta, no estará muy claro para cualquiera que lo lea qué se quiere obtener. Si pone un 1, por ejemplo, el resultado que aparecerá por pantalla será 453,6. La pregunta es ¿qué sentido tenía la entrada de datos?; en cuanto al 453,6... ¿qué significa?

Otra cosa totalmente distinta sería si se reescribe el programa de la manera siguiente:

```
10 INPUT "LIBRA";A
20 LET G=A*453.6
30 PRINT "IGUAL A";G;"GRAMOS"
```

Pruebe este programa y tendrá claro lo que queríamos explicar.

La otra razón del por qué hay que preparar la salida de datos antes de empezar a codificar el programa es que el formato de salida indica exactamente lo que se quiere hacer con el programa, cuando lo diseñe y lo codifique. Pero hay dos formas de diseñar la salida de datos. La primera es la que no tiene sentido, o sea imprimir los resultados con toda la información necesaria en pantalla, empezando en el extremo izquierdo. Quizá y por aquello de la claridad, insertaremos alguna que otra línea en blanco. Esto está bien cuando no hay que imprimir muchos resultados y cuando no le importe que la presentación sea limpia.

De todos modos, la información elegante ayuda a comprender lo que estábamos realizando, mientras que una salida de información sin mensajes falla en este aspecto. Por lo que una salida de datos elegante es preferible al caso contrario. La diferencia entre las dos estriba en la claridad. Vea los siguientes ejemplos.

```
1ØØ.ØØØ METROS EQUIVALEN A:
      MILIMETROS   1ØØØØØØØØ
     CENTIMETROS   1ØØØØØØØ
      DECIMETROS   1ØØØØØØ
          METROS   1ØØØØØ
       DECAMETROS  1ØØØØ
     HECTOMETROS   1ØØØ
       KILOMETROS  1ØØ

        PULGADAS   3937ØØ7.87
            PIES   328Ø83.99
          YARDAS   1Ø9361.33
          MILLAS   62.1371192
           NUDOS   54.ØØ5595
```

Este es un ejemplo de una salida clara de información.

1ØØØØØ METROS EQUIVALEN A:
MILIMETROS 1ØØØØØØØØ
CENTIMETROS 1ØØØØØØØ
DECIMETROS 1ØØØØØØ
METROS 1ØØØØØ
DECAMETROS 1ØØØØ
HECTOMETROS 1ØØØ
KILOMETROS 1ØØ
PULGADAS 3937ØØ7.87
PIES 328Ø83.99
YARDAS 1Ø9361.33
MILLAS 62.1371192
NUDOS 54.ØØ5595

Este, sin embargo, es un claro ejemplo de un planteamiento erróneo para la salida de información.

¿Cómo podemos preparar una salida clara y elegante? En papel, por supuesto, y antes de tocar el ordenador. Utilice papel cuadriculado, por ejemplo con cuadrados de 5 mm. de lado. En la hoja dibuje una pantalla como si fuera la del televisor. Tenga en cuenta el formato de pantalla de su ordenador. Su ZX Spectrum tiene una pantalla de 32 cuadrados de ancha por 22 de alta.

Le será muy útil fotocopiar varias de esas plantillas que haya hecho. Las necesitará.

Primero escriba en un borrador lo que tiene pensado hacer. Luego cuente las letras de cada palabra y cuando sea una frase no olvide contar los espacios en blanco. Tenga en cuenta que las palabras y las frases tienen que caber a lo largo de los 32 cuadrados que ocupa la pantalla, y cuando tenga un texto muy largo, no puede exceder de 25 líneas. Juegue con las palabras en el borrador hasta que esté satisfecho del formato conseguido. A continuación tiene un posible formato de salida al programa de conversión.

```
XXX LIBRAS EQUIVALEN A:

          GRAMOS    XXXXX
     DECAGRAMOS     XXXXX
  HECTOGRAMOS       XXXXX
  KILOGRAMOS        XXXXX
     TON. MET.      XXXXX

          ONZAS     XXXXX
        LIBRAS      XXXXX
        STONES      XXXXX
     QUARTERS       XXXXX
HUNDREDWEIGHTS      XXXXX
     TON. ING.      XXXXX

SI QUIERE HALLAR MAS EQUIVALENCIAS
TECLEE S O N Y PULSE ENTER
```

Este formato muestra claramente cómo alinear las palabras verticalmente. Cuente la longitud de la palabra más larga, en este caso «hundredweights», que tiene 14 letras. Luego asegúrese de que el resto de las palabras cuya longitud sea inferior sean rellenadas con espacios en blanco hasta completar los 14:

PRINT «l GRAMOS»
ocho espacios

## TERCER PASO: DISEÑAR EL FORMATO DE ENTRADA

Lo que acabamos de explicar para el formato de salida se puede aplicar igualmente a la entrada de datos (es menos importante que la salida, porque la entrada de datos es el punto de comienzo de un programa y no la finalidad del mismo). No hace nada malo si trabaja antes con el formato de entrada que con el programa.

Después de todo, el formato de entrada deberá indicar al usuario exactamente lo que se requiere como dato de entrada. Hay un proverbio en el mundillo de la informática resumido como «GIGO». Que significa «Garbage In, Garbage Out» («Basura de entrada, Basura de salida»). Esto significa que si damos datos erróneos a la entrada, el programa nos enviará datos erróneos de salida. Por eso un formato de entrada claro y correcto es crucial para el correcto funcionamiento del programa. Cuando especificamos que un programa ha de ser legible por cualquier usuario, no sólo hacemos referencia al formato de salida, sino que también lo hacemos al formato de entrada. Cuando cualquier persona, ade-

más del programador, encuentra dificultades en lo que se refiere a los datos de entrada, entonces el programa no es legible. Claro que los programas dedicados a problemas muy concretos no tienen por qué ser entendidos por los demás —pero si no lo entiende el especialista para quien se escribió el programa, entonces hay algo que no marcha.

El ejemplo siguiente lo aclarará un poco más:

```
10 INPUT DA
20 INPUT MO
30 INPUT YE
```

Cuando la sección de entrada de datos de un programa se ejecuta, nadie excepto el programador sabe el tipo de dato que necesita ese programa, porque el ordenador pregunta con un signo de interrogación parpadeante. Obviamente, es un diseño pobre de entrada de datos. El ejemplo 2 es totalmente distinto al primero:

```
  10 PRINT "Teclee la fecha de n
acimiento."
  20 PRINT "Primero el dia,"
  30 PRINT "luego el numero del
mes,"
  40 PRINT "y por ultimo el anno
."
  50 INPUT "DIA ";DA
  60 INPUT "MES ";MO
  70 INPUT "ANNO ";YE
  80 PRINT : REM Linea en blanco
  90 PRINT DA;" ";MO;" ";YE
```

Esta rutina de entrada de datos muestra en la pantalla una serie de mensajes y comentarios. Es una entrada bien diseñada, que hace legible el programa. Pruebe ambos ejemplos en su ordenador.

## EL MENU DE TRABAJO

Aquellos programas que ofrecen un número determinado de opciones, necesariamente han de contar con un menú de trabajo en el formato de entrada. Esto significa simplemente que en el formato de entrada se especificará claramente las diversas opciones que hay en el programa. En nuestro programa ejemplo de conversiones, hay dos opciones, porque sólo existen dos posi-

bilidades: conversiones de longitud y de peso.

Como verá más adelante, no es tan difícil escribir rutinas que dirijan el programa mediante «prompts» como se indican en el menú. Una forma simple y común de inicializar semejante opción es tecleando una letra o número que se refiera a dicha opción. En nuestro programa ejemplo la opción de conversión de medida se la puede referir mediante el número 1, por ejemplo, y la opción de conversión de peso mediante el número 2. El mensaje para la primera opción puede ser «teclee 1 y pulse RETURN para la conversión de medidas». Y algo similar pero usando el 2 para la conversión de pesos. Claro está que también hay diversas variaciones. A continuación tenemos el diseño del formato de entrada de datos para este programa.

```
   ************************************
   *                                  *
   *  CONVERSIONES   DEL  SISTEMA     *
   *        METRICO   AL   INGLES     *
   *                                  *
   ************************************

***MENU    ESCOJA  UNA    OPCION    ***

   TECLEE  EL  NUMERO  APROPIADO  Y
   PULSE  ENTER

      CONVERSION  DE  MEDIDAS  (1)

        CONVERSION  DE  PESOS  (2)

           PARAR  EL  PROGRAMA  (0)

              (TECLEE  EL  NUMERO)

L  (ESPERANDO  DATOS)
```

Ahora vea los siguientes detalles:

- La parte superior de la pantalla está ocupada por un recuadro hecho con asteriscos conteniendo el mensaje de lo que trata el programa. Este mensaje actúa como un indentificador. No está de más incluir mensajes de identificación en los formatos de entrada.
- Hay una tercera opción que es la de «PARAR EL PROGRAMA», a la que se hace referencia mediante el Ø. Aunque la forma más fácil de parar un programa es pulsando la tecla RUN/STOP o apagando el ordenador (borrando el programa totalmente), no es una solución muy legítima. Un buen programa contiene una rutina de acabado, que está incluida en el menú de trabajo.

NOTA: El esfuerzo realizado para hacer un formato de entrada lo más elegante posible mejorará la representación final del programa, dándoles un pequeño toque profesional. Procure hacerlo en todos sus programas.

## CUARTO PASO: TRABAJE CON LOS POSIBLES MODULOS DE PROGRAMACION

Este libro enseña el arte de programar de manera práctica, sin tanta teoría. De todos modos es difícil evitar consideraciones de naturaleza teórica, porque la aproximación lineal de un programa estructurado requiere alguna explicación como habrá podido comprobar en lo leído hasta aquí. Ahora, cuando veamos el tópico de programar módulos, haremos algunas consideraciones generales necesarias.

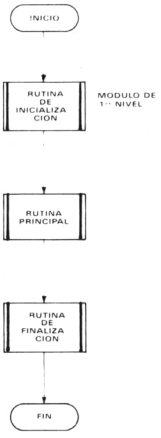
En la programación lineal es fundamental trabajar el problema antes de escribir el programa, con cuantos procedimientos sean necesarios para completar el objetivo que nos planteamos al hacer el programa. Usamos normalmente algunos nombres para estos procedimientos: subrutinas, subprogramas y módulos.

Para evitar confusiones entre estas subrutinas y aquellas a las que se accede mediante el uso del comando GO SUB, recomendamos denominarlas rutinas solamente, con el nombre que describe el proceso que realizan. Por ejemplo, rutina de conversión de peso, rutina de conversión de medidas, rutina de control principal, etcétera.

En términos generales podemos hablar de módulos de programas o solamente módulos. Podemos decir algo así como, «El siguiente módulo a escribir es la rutina de conversión de medidas.» Cada módulo debe ser individual, en el sentido de que debe funcionar independientemente del resto de los módulos. En realidad esto no es totalmente posible, porque el correcto funcionamiento del módulo depende de lo que recibe de otros módulos. Pero no es tan importante como el principio de la programación lineal; simplemente la idea es que un módulo pueda trabajar con independencia del resto de los módulos. Más adelante, en el próximo capítulo, comprenderá la base de programar módulo a módulo el programa que ya escribimos. La idea de la programación lineal es que los módulos independientes son encadenados en un orden específico, desde el primer

nivel al nivel más bajo. Haremos referencia al último párrafo del capítulo I que nos da una idea clara del principio de esta materia.

Seamos prácticos. El texto siguiente y los diagramas aclararán lo que hemos visto hasta ahora.

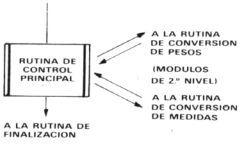



Antes que nada, cada programa estructurado tiene una parte principal. Esta puede ser un conjunto de módulos encadenados o uno solo, depediendo de la complejidad del programa. Este programa de conversión en su esencia es muy simple, por lo que su programa principal será a su vez simple. Cada parte principal tiene un módulo de comienzo y otro de fin. Entre los dos podemos colocar otro módulo que controle el programa actual. Se denominará rutina de control.

Concentrémonos en la idea general relativa a la parte principal.

El primer módulo es la Rutina de Inicialización. Este módulo puede consistir de dos partes, una sección de identificación y otra sección que inicialize todas las constantes y variables del programa. La segunda sección es de vital importancia. Según la complejidad del programa esta sección contendrá instrucciones de DIMensionamiento de variables y DEFinición de FuNciones. Todavía no hemos explicado lo que significa estas herramientas de programación en BASIC; lo veremos en el capítulo siguiente. Pero no las olvide. En cuanto a las constantes, son fáciles. Un ejemplo: si por cualquier razón necesitamos la famosa constante PI en su programa, podemos definirla en esta rutina de inicialización de la siguiente forma: PI = 3.141592654. En nuestro programa ejemplo no necesitamos inicializar funciones ni constantes.

Como ya sabrá, muchas de las variables de los programas del ZX Spectrum tienen que ser inicializados (puestos a cero). La sección de inicialización es donde tendrá lugar esta inicialización.

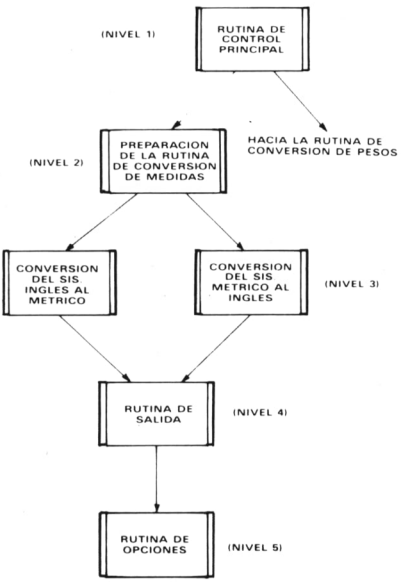


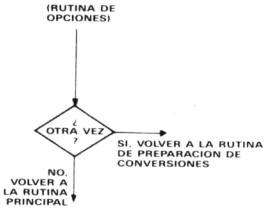

El último módulo del programa principal es, en este caso, una formalidad. Para el programa y deja el ordenador esperando a que tomemos una decisión. De todos modos, en otros programas la rutina de finalización puede ser una parte vital del programa pudiendo contener la salida final de datos. Es el segundo módulo en importancia, porque controla el flujo del programa. Recibe los datos de entrada y según esa información dirigirá el programa hacia un segundo nivel de módulos.

Cuando finaliza el flujo del programa vuelve (RETURN) al programa principal. Recuerde que una de las reglas de la programación lineal es que el control del programa siempre pasa al programa principal una vez finalizada el trabajo de la rutina.

Hasta aquí las explicaciones del programa principal. Veamos el organigrama del programa desde que se bifurca del programa principal.

Puede comprobar lo que ocurre cuando el programa bifurca hacia el siguiente nivel del programa. El flujo del programa entra en un segundo nivel de módulos, aquí la rutina de preparación de conversiones de medidas. ¿Por qué preparación? Bueno, hay dos maneras de convertir las medidas del Sistema Métrico al Sistema Inglés y del Sistema Inglés al Sistema Métrico. Este módulo se encarga de que el usuario elija la opción, el flujo del programa se encamina hacia el siguiente módulo, esto es, el tercer nivel de módulos. Aquí es donde se ejecuta la conversión. El usuario es preguntado por el número a convertir, y entonces la rutina lo convierte en el sistema adecuado y cuando se ha hecho, el flujo del programa se bifurca hacia el siguiente módulo, que a su vez se encuentra en un nivel inferior, la rutina de salida. Esta rutina produce unos buenos listados en pantalla (por lo menos, ésa es nuestra intención).






¿Finaliza aquí el programa? Si y no. Si, porque el programa ha cumplido con el objetivo: convertir números de un sistema a otro. Pero con una pasada del programa. Por eso la otra respuesta es NO, porque finalizar un programa de esta forma tan incómoda es sólo un ejemplo de programación pobre y poco elegante, ya que lo convierte en poco efectivo. Cada vez que quiera una conversión tendrá que ejecutar el programa.

Además va en contra de las reglas de la programación estructurada. Cuando el control del programa pasa a través de rutinas inferiores, siempre debe acabar en el programa principal. El programa se hace más efectivo cuando se le añade otro módulo, que le da la opción extra que le pregunta si ¿quiere ejecutarlo de nuevo? (esto es, si se desean más conversiones de pesos y medidas, depende de lo que haya escogido en el programa principal) ¿Si o no? En el primer caso, una respuesta rápida hará retornar el flujo del programa hacia la rutina de preparación de conversión que acaba de ejecutar. En el segundo caso el flujo del programa se dirigirá hacia el programa principal, o, lo que es mejor, hacia la rutina de control donde será objeto de otras preguntas.

Acabamos de discutir vagamente el flujo del programa en una ejecución. Y, como podrá ver, este programa está hecho en organigramas de módulos. De todos modos y como ya explicamos al principio del capítulo, también se puede hacer un esquema escrito de un posible organigrama del programa.

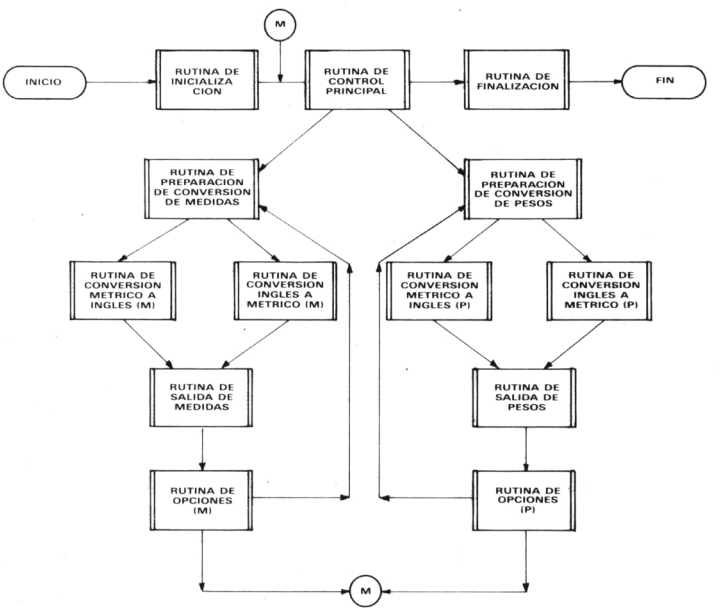
Este flujo de programa escrito con palabras se puede hacer con módulos de la misma forma que hacemos los organigramas con símbolos. Lo que aquí necesita es una habilidad lingüística en el sentido de que tiene que escribir exactamente lo que quiere hacer en el programa. Será su problema si quiere utilizar organigramas con símbolos o esquemas con palabras. En nuestra opinión lo mejor sería utilizar pequeños organigramas que le darán una rápida visión del problema pero no hay que descartar que en otros casos los esquemas puedan hacer un trabajo mejor. Cerraremos el capítulo no sólo con un organigrama general de módulos del programa completo sino que también con esquemas del programa ejecutando el mismo caso. Notará, en algunos casos, que ambos sistemas se complementan.

Para finalizar: sabemos ahora cuántos módulos son necesarios para llevar a cabo una conversión en particular.

El programa principal (nivel 1) consta de tres módu-

los. El rango de las rutinas que trabajan con la conversión de la longitud contienen, un módulo de nivel 2, dos de nivel 3, uno de nivel 4, y un módulo de nivel 5. Como la conversión de pesos requiere básicamente los mismos tratamientos es obvio que el rango de rutinas que trabajen con los pesos sean similares a las que trabajan con medidas. Esa es la razón de que el organigrama modular sea tan simétrico. Un último comentario: estos diseños preliminares que vamos a desarrollar han de ser los finales. La programación es como una partida de ajedrez: sólo los muy preparados pueden ver los movimientos con antelación. Por lo que hay una buena posibilidad de que cuando vaya a codificar el programa cambie de opinión y altere algún que otro paso. Puede ocurrir que decida empezar de nuevo. Eso no es nada preocupante. Son las cosas que hacen de la programación un trabajo altamente creativo.

Por esta razón, el programa que surge de este planteamiento inicial difiere deliberadamente en algunos niveles del diseño inicial. Aunque la idea global se mantenga.



## DIAGRAMA DE FLUJO EN PALABRAS

- Programa principal
  Comienzo
  GO TO rutina de inicializacion
  GO TO rutina de control principal
  GO TO rutina de fin del trabajo
  STOP
- Desde la rutina de control principal a la salida y más allá.

IF, si la decisión tomada en la rutina de control indica conversión de pesos GO TO rutina de preparación de pesos.

IF, si la decisión tomada en la rutina de control indica conversión de medidas GO TO rutina de preparación de medidas.

Asumir que la decisión es para la rutina de medidas.

Preparación de la rutina de conversión de medidas.

IF, si la decisión tomada en la rutina de preparación de medidas es desde el Sistema Métrico al Inglés.

GO TO entrada de datos del sistema métrico.

IF, si la decisión tomada en la rutina de preparación de medidas es desde el Sistema Inglés al Métrico.

GO TO entrada de datos del sistema inglés.

- Asumir que la decisión tomada es desde el sis. métrico al inglés.
- Entrada de datos del sistema métrico.
  Teclee los datos métricos (en metros).

Calcular los resultados de longitud en el sistema métrico estándar.

Calcular los resultados de longitud en el sistema inglés estándar.

GO TO rutina de salida.

- Rutina de salida.
  Imprime resultados de la conversión en pantalla.
  GO TO la rutina de opciones.
- Rutina de opciones.

IF repetir el proceso, vuelve a la rutina de preparación de conversión de medidas.

IF, si no se repite el proceso, vuelve a la rutina principal de control.

## CAPITULO 5:

# La codificación del programa

Este capítulo le ayudará a preparar el programa después de haberlo diseñado concienzudamente con los pasos de los capítulos anteriores. El programa se desarrollará rutina tras rutina. Cada una de ellas se desarrollará con los puntos restantes de los que se explicaron al comienzo del capítulo 4. Donde haga falta se pondrán organigramas o diagramas de palabras. Esto le dará rigor a los principios del flujo lineal de la programación estructurada.

## OBSERVACIONES GENERALES

El programa se ha comentado en abundancia con las sentencias REM, mientras que las secciones y módulos han sido separados mediante líneas de asteriscos, guiones, etc. Esto se ha hecho para reforzar la idea de programación modular (programas que consisten en bloques de procedimientos, subprogramas, subrutinas o como lo quiera llamar).

En la mayoría de los casos, las sentencias REM, que explican las partes ejecutables del programa se han situado con anterioridad a esas partes. La legibilidad se ve reforzada por el continuo uso de líneas en blanco (teclee un número y seguidamente los dos puntos, :). También podemos tener una línea en blanco tecleando el número y seguidamente poner REM.

En puntos estratégicos del programa encontrará el comando «CLS». Este limpiará la pantalla.

Hay sentencias REM en este programa que realmente se las puede considerar como supérfluas. Pero se han puesto para hacer el programa lo más comprensible posible para el novato en este campo.

## PRIMERO EL PROGRAMA PRINCIPAL

(Vea el capítulo 4, puntos 6, 7 y 8).

Es esencial dentro de los esquemas de la programación lineal, empezar el programa en la rutina de más alto nivel y en consecuencia trabajar hacia abajo. La parte principal es la rutina del primer nivel, y como se estableció en su momento, consiste de tres módulos.

Desarrolle por completo esta parte principal y compruebe su correcto funcionamiento, antes de hacer el siguiente módulo (punto 8: Procederá con el nivel siguiente solamente cuando la rutina actual funcione y esté verificada en su totalidad).

Consideremos los módulos individualmente.

## EL MODULO PRINCIPAL DE CONTROL

Esta es la rutina verdaderamente importante del primer nivel, ya que dirige el flujo del programa hacia la rutina correcta de proceso.

Así es como se ha preparado (diagrama del flujo del programa en palabras):

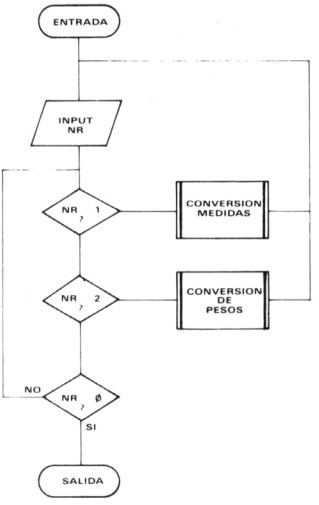
- Entrada en la rutina de control principal.
- Situar en pantalla las opciones.

(1) para conversiones de longitud (medidas).
(2) para conversiones de pesos.
(Ø) para parar el programa.

- IF, si la entrada es igual a 1 entonces (THEN) ir (GO TO) a la Rutina de Preparación de Conversión de Medidas y seguidamente volver (RETURN) a la Rutina de Control Principal.
- IF, si la entrada es igual a 2 entonces (THEN) ir (GO TO) a la Rutina de Preparación de Conversión de Pesos y seguidamente volver (RETURN) a la Rutina Control Principal.
- IF, si la entrada es igual a Ø entonces (THEN) ir (GO TO) a la Rutina de Finalización.

(si se pulsa alguna opción que no esté especificada, bifurca a la primera condición).

Para dejarlo más claro todavía, a continuación está el organigrama:




```
 570:
 580 REM **rutina principal**
 590:
 600 PRINT : REM linea en blanco
 610:
 620 PRINT "  teclee la opcion q
ue desee"
 630 PRINT "   y pulse ENTER:"
 640 PRINT
 650 PRINT "   para convertir me
didas (1)"
 660 PRINT
 670 PRINT "   para convertir pe
sos (2)"
 680 PRINT
 690 PRINT "   para finalizar el
 prog. (0)"
 700 PRINT
 710 PRINT "   Teclee un numero.
"
 720:
 760 INPUT "            ";nr
 770:
 790 IF nr=1 THEN GO TO 960
 800 IF nr=2 THEN GO TO 1130
 820 IF nr=0 THEN GO TO 870
 830:
```

De manera que lo que ocurre en esta rutina de control es:

Cuando se ponen en pantalla las opciones (líneas 62Ø-76Ø) el programa espera en la línea 76Ø la en-

trada (INPUT) de un número. No es ninguna particularidad que se hayan dejado tantos espacios en blanco entre el comando INPUT y la variable NR. Esto asegura una buena representación por pantalla al aparecer el cursor en el centro de esta, preguntando al usuario por la opción elegida. La línea 79Ø contiene el primer test condicional. IF NR = 1 THEN GO TO 96Ø, que aparentemente es el comienzo de la rutina de Preparación de Conversión de Medidas. De todos modos, cuando codifique esta parte de programa, no sabrá dónde empezará esta rutina. Para asegurarse que funcione la rutina principal de control teclee lo siguiente:

79Ø IF NR = 1 THEN TO GO 9999

y a continuación:

9999 PRINT «PREP.L.CONV.RUTINA»: STOP

Lo que ocurre ahora es que cuando se ejecute la rutina principal del programa, este no se venga abajo («crash») en la línea 79Ø, porque el programa al menos se dirige hacia algún sitio (GO TO 9999). Haga lo mismo con la líena 8ØØ:

8ØØ IF NR = 2 THEN GO TO 9998

Más tarde, cuando haya acabado con las rutinas del segundo nivel en cuestión, tendrá que alterar las líneas 79Ø y 8ØØ, de manera que dirijan el flujo del programa hacia la zona adecuada (y borre las líneas 9999 y 9998).

La línea 82Ø es una medida precaución. Ocurre con frecuencia que se teclean números equivocados accidentalmente. La línea 82Ø GO TO 76Ø se anticipa a esto. Después de todo con cualquier número que no sea el Ø ni el 1 ni el 2, los test serán falsos y el programa se dirigirá hacia la rutina de finalización. El organigrama aclarará esto.

## RUTINA DE FINALIZACION

El siguiente programa habla por sí solo.

```
840:
850 REM **rutina  final**
860:
870 CLEAR
880 PRINT "    fin del programa"
890 GO TO 9000
900:
910 REM ======================
920 REM ======================
```

Limpia la pantalla y todas las variables, sacando un mensaje por pantalla: «FIN DE PROGRAMA» y READY. El programa se para en la línea, 9ØØØ, porque no tiene ningún sitio a donde ir. Como puede ver, la línea 89Ø, que manda el programa a la línea 9ØØØ, podía haberse sustituido por 89Ø STOP. Esto es menos elegante ya que el ordenador le enviará un mensaje de que el programa se ha parado (STOP). El STOP implica una parada temporal y no el fin del programa. 9ØØØ es la última línea del programa donde acaba de verdad. En casos como este, la rutina de finalización es muy simple. Según la complejidad de los programas las rutinas de finalización (o módulos) pueden ser complejas a su vez. Pueden contener importantes rutinas de salida.

## RUTINAS DE INICIALIZACION

```
 100 CLS : REM limpia  pantalla
 110:
 120 REM ...rutina principal...
 131:
 132 REM seccion de identificaci
on
 133:
 140 PRINT "   ******************
***********"
 150 PRINT "   *
            *"
 160 PRINT "   * conversion del s
is. ingles*"
 170 PRINT "   * al sis. metrico
y vice-    *"
 180 PRINT "   * versa
            *"
 190 PRINT "   ******************
***********"
 200:
 210 REM autor - edward barton
 220 REM escrito en 9-2-84
 230 REM version final
 240:
 245 REM calc. centimetros
 250 REM descripcion  programa
 251 REM este programa convierte
 252 REM las unidades estandard
 253 REM de pesos y medidas
 254 REM del sistema metrico
 255 REM y del sistema ingles
 260:
 270:
 280 REM ----------------------
```

(Ver capítulo IV, punto 11).

Aunque no sea una parte esencial del programa, siempre es conveniente inicializar un programa con una sección de identificación. Por aquello de mejorar la documentación, también es importante incluir una sección que describa las variables utilizadas en el programa.

Una Rutina de Inicialización normalmente contiene una parte esencial del programa. Como ya se vió (capítulo IV) puede contener las definiciones de las constantes, definición de funciones y el dimensionamiento de las cadenas (trataremos las cadenas en el capítulo siguiente). No encontrará otra sección como ésta en la Rutina de Inicialización, porque el programa completo es tan simple que tantas medidas especiales no hacen falta.

```
285:
290 REM descripcion  variables
300:
310 REM nr entrada del menu
320 REM a$ & b$ variables
330 REM kg contiene kilogramos
340 REM lb contiene libras
350 REM mt contiene metros
360 REM ft contiene pies
370 REM gr contiene gramos
380 REM dg contiene decagramos
390 REM hg contiene hectogramos
400 REM tn contiene toneladas
410 REM ou contiene onzas
420 REM sn contiene stones
430 REM qu contiene quartes
440 REM hw contiene hundredwht
450 REM it contiene ton. ing.
460 REM mm contiene milimetros
470 REM cm contiene centimetros
480 REM dm contiene decimetros
490 REM da contiene decametros
500 REM hm contiene hectometros
510 REM km contiene kilometros
520 REM in contiene pulgadas
530 REM ya contiene yardas
540 REM ml contiene millas
550 REM kn contiene nudos
555:
560 REM ======================
```

En cuanto a la sección de Identificación y la sección que describe las variables, el autor de este programa comenzó codificando el programa con la sección de Identificación y terminó describiendo las variables. (Pertenece al punto 9 del capítulo IV, que dice así: «Deje los refinamientos para el final del programa»). Después de todo, nunca se sabe a ciencia cierta el número de va-

riables que vamos a utilizar en el programa de antemano. Por lo que el programador ha reservado la memoria suficiente entre las líneas 3ØØ y 58Ø (comienzo del rutina de control principal) y las ha rellenado con sentencias REM que describen cada variable.

NOTA: Tenga en cuenta que el Spectrum tiene 16 Kbytes de memoria, en su versión más baja. Si no tiene usted uno de 48 Kbytes, es una cantidad considerable pero se puede quedar sin ella antes de que se dé cuenta, si utiliza demasiadas sentencias REM. En estos casos tendrá que sacrificar alguno de ellos. La descripción de las variables puede ser la primera en eliminarse.

## RUTINA DE PREPARACION DE CONVERSION DE PESOS Y MEDIDAS

```
 930:
 940 REM *rutina conv. med.*
 950:
 960 CLS
 970:
 980 PRINT " conversion medidas"
 990:
1000 PRINT : REM linea en blanco
1010:
1015 REM mensajes de pantalla
1017 REM subrutinas
1020 GO SUB 3070
1030:
1040 INPUT a$
1050 IF a$="m" THEN GO TO 1680
1060 IF a$="i" THEN GO TO 1860
1070 GO TO 1040
1080:
1090 REM ----------------------
1100 REM *rutina conv. pesos*
1120:
1130 CLS
1140:
1150 PRINT " conversion pesos"
1160:
1170 PRINT : REM linea en blanco
1180:
1190 GO SUB 3070
1200:
1210 INPUT a$
1220 IF a$="m" THEN GO TO 1320
1230 IF a$="i" THEN GO TO 1500
1240 GO TO 1210
1250:
1260 REM ======================
```

Ambas rutinas están en el segundo nivel y básicamente tienen los mismos pasos. Siguen una estructura similar de entrada de datos y toman decisiones similares a las existentes en la rutina de control principal. Aunque hay una subrutina utilizada por estos dos módulos: es la subrutina 3Ø7Ø y se hace referencia en las líneas 1Ø2Ø y 119Ø.

Es una subrutina muy simple, que evita trabajar el doble. En ambos casos hay un mensaje en la pantalla que requiere del usario una respuesta ya sea una I o una M, si él/ella quiere hacer la conversión del sistema inglés al métrico o del métrico al inglés respectivamente. La subrutina 3Ø7Ø es la que genera este mensaje. Encontrará esta subrutina al final del listado del programa.

Para que no tenga muchas dudas, aquí tiene el diagrama por palabras:

- Limpiar la pantalla.
- Escribir el nombre de la Rutina (Conversión de Medidas).
- Escribir las opciones: «METRICO A INGLES, PULSE M Y ENTER.»
  «INGLES A METRICO, PULSE I Y ENTER.»
- IF, si la entrada es igual a M ir (GO TO) Inicalización Métrica al Inglés.
- IF, si la entrada es igual a I ir (GO TO) Inicialización Inglés al Métrico.
- IF, si se ha pulsado alguna otra tecla vuelve al test 1º.

NOTA: aparentemente tenemos la primera diferencia en cuanto a la idea general, que dice que desde este módulo el programa ha de bifurcarse a un módulo de salida. Parece ser que hicieron falta más módulos extras. Como se indicó en los capítulos anteriores, esta diferencia con el plan original se ha hecho deliberadamente. Pero insistimos que la idea general se debe mantener, a lo largo del programa.

```
1270:
1280 REM *rut. pesos met/ing*
1290:
1300 REM kg-unidad standard
1310:
1320 CLS : REM limpiar pantalla
1330:
1340 PRINT "teclee cantidad kg"
1350 PRINT "y pulse ENTER"
1360 INPUT "kilogramos?";kg
1390 PRINT kg;:
1400:
1410 PRINT "  kilogramos equival
en a"
```

```
1420 REM ir (GO TO) rutina de
1425 REM calculo y de entrada
1426 GO TO 2060
1430:
1440 REM ----------------------
1450:
1460 REM *rut. medidas met/ing*
1470:
1480 REM libra-unidad standard
1490:
1500 CLS
1510:
1520 PRINT "teclee cantidad de "
1525 PRINT "lb y pulse ENTER"
1540 INPUT " libras ";lb
1542:
1545 REM convierte lb. a kg.
1550 LET kg=lb*0.4536
1560:
1570 CLS
1580 PRINT lb;:
1590 PRINT "  libras equivalen a
"
1600 GO TO 2060
1610:
1620 REM ======================


1630:
1640 REM *inicio long. met/ing*
1650:
1660 REM metro-medida standard
1670:
1680 CLS
1690:
1700 PRINT "teclee cantidad de"
1705 PRINT "metros y pulse ENTER
"
1710:
1720 INPUT " metros ";mt
1730:
1740 CLS
1750:
1760 PRINT mt;:
1770 PRINT " metros equivalen a"
1780 GO TO 2450
1790:
1800 REM ----------------------
1830:
1840 REM pies-medida standard
1850:
1860 CLS
1870:
1880 PRINT "teclee cantidad de"
1885 PRINT "pies y pulse ENTER"
1890 PRINT
1900 INPUT " pies ";ft
1905 REM convertir ft. a cm.
1910 LET cm=ft*30.48
1915 REM convertir cm. a mt.
1920 LET mt=cm/100
```

```
1930:
1940 CLS
1950:
1960 PRINT ft;:
1970 PRINT " pies equivalen a"
1972:
1975 REM ir (GO TO) rutina de ca
lculo y salida
1980 GO TO 2450
1990:
2000 REM ====================
```

## 4 RUTINAS DEL TERCER NIVEL

Ahora tenemos cuatro rutinas del tercer nivel:

- Inicialización del sistema Métrico al Inglés de pesos.
- Inicialización del sistema Inglés al Métrico de pesos.
- Inicialización del sistema Métrico al Inglés de medidas.
- Inicialización del sistema Inglés al Métrico de medidas.

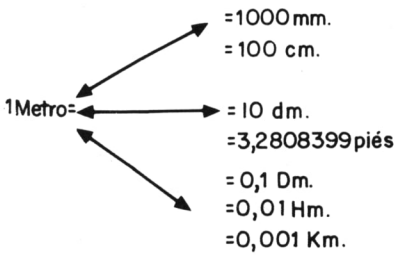
Todas tienen una cosa en común; se requiere una entrada de datos (INPUT), en este caso un número a convertir. La primera de las cuatro saca por pantalla el mensaje siguiente: «TECLEE NR DE KG Y PULSE ENTER.». El número tecleado se almacenará en la variable KG, luego imprimirá (línea 139Ø) el contenido de esta variable , seguido de las palabras «kilogramos que equivalen». La línea 139Ø se encarga de esto. PRINT KG (punto y coma seguido de dos puntos) sitúa la sentencia PRINT «KILOGRAMOS QUE EQUIVALEN» después del contenido de KG. A continuación el programa procede a la rutina para calcular el peso, que es al mismo tiempo la rutina de salida.

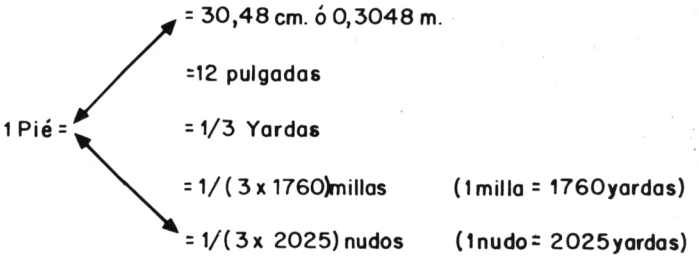
Observe que las cuatro rutinas de tercer nivel siguen el mismo principio en lo que se refiere al flujo de los programas.

Pero ponga especial atención en el caso de dos de estas rutinas (Del sistema Inglés al Métrico de pesos y de medidas) una de las conversiones básicas se ha llevado a cabo, libras son convertidas en kilogramos y pies convertidos en metros.

Es aquí donde en verdad comienza la labor del programa: trabajar con los números para conseguir la salida correcta. A estas alturas, el programador ha de pensar con lógica y ha de procurar encontrar el camino más idóneo que se le ocurra (no se sorprenda si encuentra una solución mejor cuando haya acabado el programa, éste o cualquier otro: es una de las razones por la que muchos programas son actualizados con el tiempo). Como verá, su ordenador es una máquina tonta. Codificar erróneamente el programa le causarán resultados igual de equivocados (recuerde cuando comentamos los errores lógicos). O sea que el consejo general es pensar detenidamente lo que hay que hacer. Asegúrese de que los cálculos son los correctos. Cuando haya codificado el programa, haga de ordenador y siga los pasos como lo haría él, antes de teclear el programa. Jugar a ser el ordenador significa que tendrá que probar todos ls casos como si de una máquina se tratara, y comprobar que los resultados son los que deseamos. Una calculadora será muy útil en situaciones cruciales como éstas.

Pero ¿cómo se ha resuelto el problema? La finalidad del programa es producir la salida con las conversiones en ambos sistemas, Métrico o Inglés, ya sea de pesos y medidas. Esto significa que una entrada (INPUT) en metros se tiene que convertir no solamente al estándar Inglés sino también a otras medidas estándar del sistema Métrico. Lo mismo para la entrada en pies. No solamente hay que convertirla en metros sino que también en otras medidas estándar del sistema Inglés. Esto significa que ambas conversiones tendrán el mismo formato de salida, lo que simplificará las tarea considerablemente. Podemos ir más allá en la simplificación del problema —todas las conversiones se pueden derivar de unas conversiones básicas. Vea la figura siguiente:






NOTA: tenga en cuenta la conversión Metros-Pies y vea de nuevo la figura en conjunto. Podemos deducir la siguiente afirmación: CONOCIENDO UN VALOR, CONOCEMOS LOS DEMAS. Según esto, podrá escribir una rutina de cálculo que obtenga paso a paso todos los valores posibles de uno dado. Este principio se ha mantenido a través de ambas corrientes —conversión de medidas y de pesos.

Estudie estas secciones detenidamente; sugerimos que haga algunos cálculos aritméticos por su cuenta, para saber lo que hacen estos módulos.

## LAS RUTINAS DE SALIDA

```
2010:
2020 REM *rutina de calculo y
2025 REM salida de pesos
2030:
2040 REM unidades metricas peso
2050:
2055 REM calc. gramos
2060 LET gr=kg*1000
2065 REM calc. decagramos
2070 LET dg=kg*100
2075 REM calc. hectogramos
2080 LET hg=kg*10
2085 REM calc. kilogramos
2090 LET kg=kg*1
2095 REM calc. ton. metricas
2100 LET tn=kg/1000
2110:
2120 REM unidades ing. peso
2130:
2135 REM calc. libras
2140 LET lb=kg/0.4536
2145 REM calc. onzas
2150 LET ou=lb*16
2155 REM calc. stones
2160 LET sn=lb/14
2165 REM calc. quarters
2070 LET dg=kg*100
2075 REM calc. hectogramos
2080 LET hg=kg*10
2085 REM calc. kilogramos
2090 LET kg=kg*1
2095 REM calc. ton. metricas
```

```
2100 LET tn=kg/1000
2110:
2120 REM unidades ing. peso
2130:
2135 REM calc. libras
2140 LET lb=kg/0.4536
2145 REM calc. onzas
2150 LET ou=lb*16
2155 REM calc. stones
2160 LET sn=lb/14
2165 REM calc. quarters
2170 LET qu=lb/28
2175 REM calc.hundredweights
2180 LET cw=lb/112
2185 REM calc. ton. ingl.
2190 LET it=lb/2240
2200:
2210 REM salida de la conversion
2215 REM del peso
2220:
2230 PRINT
2240 PRINT "          gramos ";gr
2250 PRINT "     decagramos ";dg
2260 PRINT "    hectogramos ";hg
2270 PRINT "     kilogramos ";kg
2280 PRINT "   ton. metricas";tn
2290 PRINT
2300 PRINT "           onzas ";ou
2310 PRINT "          libras ";lb
2320 PRINT "          stones ";sn
2330 PRINT "        quarters ";qu
2340 PRINT "hundredweights ";cw
2350 PRINT "   ton. inglesa ";it
2360:
2370 GO TO 2820
2380:
2390 REM ----------------------
```

Como podrá comprobar, ambas rutinas de salida, que son del cuarto nivel, son en verdad rutinas de cálculo. No hay mucho más que comentar acerca de ellas, ya que de por sí son bastante claras.

```
2400:
2410 REM *rutina de calculo y de
2415 REM salida*
2420:
2430 REM unidades metricas de
2435 REM medida
2440:
2445 REM calc. de milimetros
2450 LET mm=mt*1000
2455 REM calc. de centimetros
2460 LET cm=mt*100
2465 REM calc. de decimetros
2470 LET dm=mt*10
2475 REM calc. de metros
2480 LET mt=mt*1
```

```
2485 REM calc. de decametros
2490 LET da=mt/10
2495 REM calc. de hectometros
2500 LET hm=mt/100
2505 REM calc. de kilometros
2510 LET km=mt/1000
2520:
2530 REM unidades inglesas de
2531 REM longitud
2540:
2545 REM calc. de pulgadas
2550 LET in=cm/2.54
2555 REM calc. de pies
2560 LET ft=in/12
2565 REM calc. de yardas
2570 LET ya=ft/3
2575 REM calc. de millas
2580 LET ml=ya/1760
2585 REM calc. de nudos
2590 LET kn=ya/2025
2600:
2610 REM salida long. inglesa
2620:
2630 PRINT
2640 PRINT " milimetros ";mm
2650 PRINT "centimetros ";cm
2660 PRINT " decimetros ";dm
2670 PRINT "      metros ";mt
2680 PRINT " decametros ";da
2690 PRINT "hectometros ";hm
2700 PRINT " kilometros ";km
2710 PRINT
2720 PRINT "    pulgadas ";in
2730 PRINT "        pies ";ft
2740 PRINT "      yardas ";ya
2750 PRINT "      millas ";ml
2760 PRINT "       nudos ";kn
2770:
2780 GO TO 2940
2790:
```

## LAS RUTINAS QUE GENERAN LAS OPCIONES

```
2800 REM ======================
2810:
2820 REM *rutina peso (opcion)*
2830:
2850 GO SUB 3145
2860:
2870 INPUT b$
2880 IF b$="y" THEN GO TO 1130
2890 IF b$="n" THEN GO TO 3200
2900 GO TO 2870
2910:
2920 REM ----------------------
```

```
2930:
2940 REM *rutina medida (opcion)
2950:
2970 GO SUB 3145
2980:
2990 INPUT b$
3000 IF b$="y" THEN GO TO 960
3010 IF b$="n" THEN GO TO 3200
3020 GO TO 2990
3030:
3040 REM ======================
3060:
3070 REM *mensajes de pantalla*
3080:
3090 PRINT "metrico a ingles,"
3095 PRINT "teclee m y pulse ENT
ER"
3100 PRINT
3105 PRINT "ingles a metrico,"
3110 PRINT "teclee i y pulse ENT
ER"
3120 RETURN
3135 REM *mensajes de pantalla*
3140:
3145 PRINT : PRINT
3150:
3155 PRINT "mas ? teclee s/n"
3156 PRINT "pulse  ENTER"
3160 RETURN
3170:
3190 REM *RETURN de la subrut.*
3195:
3200 CLS
3205 REM *volver a la rutina pri
ncipal*
3210 GO TO 600
9000 PRINT "listo"
```

Ya hemos indicado en nuestro organigrama general del programa completo que al usuario hay que ofrecerle varias opciones cuando el programa haya completado una ejecución. La opción en este caso sería ¿quiere ejecutarlo de nuevo para hacer alguna conversión de pesos o medidas? Si la respuesta es sí, entonces el programa bifurcará al punto de entrada de esa rutina en particular —ya sea de pesos o de medidas— dependiendo de en que esté en ese momento. De lo contrario, esto es, si no quiere hacer más conversiones o simplemente quiere finalizar el programa, entonces el control del programa pasará a la rutina principal —en este caso la rutina de Control Principal. Estas dos rutinas de opciones hablan por sí solas.

NOTA: las dos opciones tienen un punto de salida (línea 32ØØ), un ENTER al programa principal. Limpie

la pantalla y luego, cuando el programa entre en la rutina principal, el menú principal se visualizará en pantalla y podrá empezar de nuevo el proceso completo, pulsando el 1 ó el 2. O si quiere finalizar, pulse el Ø. Y el circulo se cerrará.

En cuanto a los dos comentarios visualizados en pantalla, no tienen más significado que lo que dicen.

## AHORA LE TOCA A USTED

Acaba de ver un programa escrito totalmente de acuerdo con las reglas de la programación estructurada y la realización lineal de programas. El resultado es un programa que, solo con mirarlo, es fácil de entender. Cuando por casualidad vea este programa dentro de unos veinte años, no tendrá problema alguno en descifrar de qué se trataba.

Pero ahora le toca a usted. Teclee este programa en base a los módulos, y compruébelo cada vez que acabe con uno. Claro que el programa parará (el ordenador pondrá un mensaje de error en la pantalla, pero no se preocupe por él). Utilice la técnica ya descrita bajo el párrafo de rutina principal. Donde un módulo se dirija a otro que no exista en ese momento, escriba unas cuantas líneas en su lugar. Por ejemplo, vea las líneas 121Ø y 124Ø en la rutina de preparación de Pesos. En estas líneas el programa se supone que ha de bifurcar hacia otro módulo pero cuando ese módulo no exista escriba en la línea 122Ø algo parecido a esto:

122Ø IF A$ = «M» THEN STOP

En este caso cuando ejecute el programa, en el momento de pulsar una «M» se parará en esa rutina. O lo puede escribir de la siguiente manera:

122Ø IF A$ = «M» THEN GO TO 9999

La línea 9999 se puede utilizar como una línea de programa sustituyendo el módulo que no sea desarrollado hasta ese momento. La línea 9999 puede ser la siguiente:

9999 PRINT «MODULO SIN ACABAR»

Esta línea de programa, o módulo de programa, se puede utilizar cada vez que quiere probar un programa y no se han acabado de escribir los módulos.

## COMETIENDO ERRORES

Cuando no se tiene experiencia escribiendo a máquina (pero incluso cuando se es un buen mecanógrafo) se cometarán errores causando que el programa se venga abajo (crash). El ordenador visualizará mensajes de error por pantalla. No se asuste en estos casos. Es parte del arte de la programación. De todos modos si ha trabajado con todos los ejercicios de los capítulos anteriores ya estará acostumbrado a los mensajes de error. Como se ha comentado, normalmente se deben a errores de tipo mecanográfico. A lo mejor se le olvidó de teclear un punto y coma, o tecleó un punto y coma en vez de dos puntos. A lo mejor se olvidó de teclear las comillas, o desea teclear los comandos letra a letra en vez de utilizar las clásicas teclas que escriben el comando entero de una vez, etc. Tómese el tiempo que necesite en corregir los errores y corríjalos.

## ¿Y AHORA QUE?

Asumamos que acaba de teclear el programa entero sin un solo error, y lo ha ejecutado correctamente, entonces se preguntará ¿Y ahora qué?

Esto ya es cosa de cada uno. Ya está listo para continuar por su cuenta. Hasta el momento le hemos llevado de la mano, pero de ahora en adelante tendrá que recurrir a su propia inventiva y creatividad. Estas son partes esenciales de la psicología de un programador. Si cree que todavía no posee esos requisitos, sólo podemos decirle una cosa: desarróllelos. Tenga en cuenta que su herramienta para la programación —su ordenador— es un aparato muy paciente. Nunca se aburrirá, ni se enfadará cuando haga algo mal. Le indicará una y mil veces lo que está haciendo mal. También tiene pocas posibilidades de poder destruirlo. Lo único que le puede ocurrir es que se bloquee, lo que significa simplemente que el pulsar teclas erróneas le ha hecho entrar en un bucle sin fin dentro de la estructura electrónica del aparato. Sólo tendrá que apagar el ordenador y encenderlo de nuevo. Volverá a ser normal.

Una última palabra, sobre el programa de conversión y el hecho de que se ha desarrollado en la base de la programación lineal. Este hecho hace posible que se

pueda extender el programa. Es relativamente sencillo añadir módulos extras que, por ejemplo, hagan conversiones del sistema Métrico al Inglés en cuanto a volumen y viceversa. Naturalmente podrá añadir un módulo que convierta la temperatura. Lo importante es añadir algunas opciones y probarlos en la rutina de control principal. Por ejemplo:

- Conversión de Volúmenes (3).
- Conversión de Temperaturas (4).

Podrá añadir a su vez dos test condicionales en las líneas 79Ø y 82Ø. De manera que este puede ser su próximo trabajo. Pero, de nuevo, es su trabajo el encontrar el camino.

## MANTENGA EL DESARROLLO LINEAL

Cualquier cosa que haga, cualquier programa brillante que desarrolle, procure siempre mantener el desarrollo lineal y las reglas de la programación estructurada, según se explicó en este y demás capítulos. Sólo sacará beneficio de ello. Sus programas serán claros, eficientes, fácilmente accesibles para actualizaciones y demás. Y si alguna vez ha pensado en convertirse en programador profesional, lo que ha aprendido aquí le será útil, le facilitará el aprendizaje de otros lenguajes de programación, como el PASCAL.

Los siguientes capítulos le introducirán a los comandos del BASIC y funciones que hasta ahora no hemos discutido.

## CAPITULO 6:

# Otras Herramientas del BASIC

En los capítulos anteriores se habrá familiarizado con las siguientes herramientas del lenguaje de programación BASIC: INPUT, PRINT, variables, operadores aritméticos, funciones de interrupción, (. , : ; " "), REM, IF-THEN, GO TO, GO SUB-RETURN.

Con este limitado juego de instrucciones del BASIC ya habrá codificado programas que, aunque simples, funcionan. En este y el siguiente capítulo veremos más comandos, instrucciones, funciones, etc, del BASIC. Y ahora que se ha familiarizado con la programación y con la jerga del programador, le introduciremos más términos nuevos.

## VARIABLES ALFANUMERICAS

Ya se ha cruzado con las variables en cadena (o variables alfanuméricas). Vea el ejemplo en las líneas 1Ø5Ø y 1Ø6Ø del programa de conversión (capítulo V):

```
1Ø5Ø IF A$ = "M" THEN GO YO 168Ø
1Ø6Ø IF A$ = "A" THEN GO TO 186Ø
```

Observe el signo del dólar después de la A, y observe sobre todo las comillas de las letras M e I. Cuando añadimos el signo dólar a una variable la convertimos en una variable de cadena. También denominada variable alfanumérica, porque puede contener caracteres alfabéticos y numéricos. Pero tenga en cuenta que esas variables sólo pueden almacenar datos que están entre comillas.

Por ejemplo:

```
LET A$ = "Juanito"
LET X$ = "AB-12"
LET N$ = "LONDRES"
LET K$ = "123456789Ø"
```

Si se olvida de colocar los datos entre comillas, su ZX Spectrum reaccionará inmediatamente con un mensaje de error (un signo de interrogación y un cursor parpadeante). Teclee lo siguiente:
LET A$ = 1 y pulse ENTER.

Sin embargo,
LET A$ = "1" será aceptado.

IMPORTANTE: en contraste con otros ordenadores, el BASIC del ZX Spectrum no acepta variables en cadena con más de un carácter antes del signo dólar. Por lo que AA$, X1$, XY$, etc. no se admiten.

## AND, OR, NOT

Estas tres palabras clave se denominan operadores lógicos y se utilizan junto con la combinación IF-THEN. Como ya sabrá, la combinación IF-THEN prerara una condición a probar. Cuando añada los operadores lógicos AND, OR, NOT, podrá hacer más de un test. en una sentencia.

Por ejemplo:
IF A = 5 AND B = 1Ø AND C = 15 THEN LET D = 2Ø

Esta sentencia de programa sólo se ejecutará cuando las tres condiciones se verifiquen: si A = 5 y B = 1Ø pero C = 6, entonces la evaluación global de la sentencia sería falsa y esta línea de programa sería ignorada por el ordenador.
IF A = 3 OR A = 7 OR A = 13 THEN STOP

En este caso la línea se ejecutará siempre y cuando sea cierto uno de los valores de A, ya sea 3, 7 ó 13.
IF NOT A = 5 AND B = 7 OR C = 1Ø THEN PRINT "ESTO ES UNA TONTERIA"

Aquí el valor de A puede ser cualquier número excepto el 5 y cuando B = 7 OR C = 1Ø la condición general se cumple y la línea se ejecutará. De manera que si A = 4 y B = 4 pero C = 1Ø, entonces se cumpliría la condición y se ejecutaría la línea.

Teclee las siguientes líneas y ejecútelas:

```
10 INPUT "A";A
20 INPUT "B";B
30 INPUT "C";C
40 REM Colocar aqui alguna lin
ea de las anteriores
50 GO TO 10
```

Sabiendo esto podrá examinar y teclear este simple juego de adivinanzas:

```
  10 PRINT "TECLEE CUALQUIER NUM
ERO"
  20 LET CN=1: REM CONTADOR DE I
NTENTOS
  30 INPUT A: REM TECLEE CUALQUI
ER NUMERO
  40 IF A=45 OR A=72 OR A=93 THE
N GO TO 100
  50 LET CN=CN+1
  60 GO TO 30
 100 PRINT "ACERTASTE!!"
 110 PRINT "CONSEGUISTE ADIVINAR
 EL NUMERO"
 120 PRINT "EN ";CN;" INTENTOS"
```

## EL SIMBOLO + COMO OPERADOR ALFANUMERICO

Al igual que con el símbolo (=), el símbolo de suma (+) tiene una doble función (= puede significar igual pero también significa sustituir por). No sólamente es un operador aritmético que suma A a B, sino que también puede concatenar variables alfanuméricas. En otras palabras, añadiendo una cadena a otra podemos crear una nueva cadena. Por ejemplo:

```
10 LET A$="LUN"
20 LET B$="ES"
30 LET C$=A$+B$
40 PRINT C$
```

El resultado por supuesto será LUNES. El comando PRINT (A$ + B$) dará el mismo resultado.

## CLEAR

La función de esta palabra clave es limpiar (borrar) todas las variables y limpiar la pantalla. No siempre es un comando útil. Tiene que saber lo que está haciendo cuando lo quiere utilizar. El manual de programación del ZX Spectrum especifica alguna cosa más.

## CLS

Este comando es muy útil. Limpia la pantalla y desecha información innecesaria. Como pudo comprobar en el programa de conversiones, se utiliza para poder obtener pantallas bien dispuestas y elegantes. Recuerde que el comando INPUT automáticamente limpia la pantalla, cada vez que lo utiliza en sus programas.

## DATA Y READ

DATA es una instrucción no ejecutable. Se le puede llamar indicador, porque indica dónde hay datos para procesar. Una sentencia DATA se parece a lo siguiente: 1Ø DATA 5.7,8.3,1ØØØ, 45, "LONDRES", "ASD", 1Ø.22,Ø,Ø,999

Cada elemento de la data está separado por una coma del anterior item. Tenga en cuenta que al final de la sentencia no se ha puesto ninguna coma. Observe a su vez que hay datos numéricos y alfanuméricos. Está permitido mezclar datos numéricos con alfanuméricos, pero no es recomendable. Las líneas con tipos mezclados de datos normalmente dan error.

¿Cuál es la función de todo esto? Las DATAS están ahí para ser leídas (READ). Las líneas de datos como las del ejemplo no tienen ningún sentido cuando no hay comandos READ que las lean en algún lugar del programa.

El ejemplo siguiente aclarará el sentido que tiene la combinación de los comandos DATA-READ. Teclee el siguiente programa y ejecútelo:

```
10 READ A
20 IF A>25 THEN PRINT A
30 IF A=999 THEN GO TO 50
40 GO TO 10
50 STOP
60 DATA 60,15,22,40,68,24,31,4
4,666,721,999
```

Como verá, este programa imprime todos los números de la DATA que sean mayores de 25. El programa es un bucle y en cada ciclo se lee (READ) un nuevo item de la serie de las DATAS, uno detrás de otro.

También observará que la línea de DATA está situada al final del programa. Se dice que las sentencias DATA se pueden colocar en cualquier lugar el programa,

pero es una buena práctica de programación el situarlas al comienzo o al final del programa. Una colección de items de datos se pueden denominar bloques de DATA.

Ahora, un pequeño ejercicio (ver el programa anterior):

Borre la línea 3Ø (teclee 3Ø y pulse ENTER). Ahora ejecute el programa. comprobará que el programa seguirá listando los números mayores de 25. Sin embargo, el programa finalizará con un mensaje de error:
E OUT OF DATA, 1Ø:1

Esto ocurre porque se ha convertido el programa en un bucle sin fin al haber quitado la línea 3Ø. No hay manera de salirse del bucle. De todos modos, después de leer los 11 items de la data, el programa no tiene más que leer y se para (crashes).

Hay algunas reglas que tendrá que tener en cuenta a la hora de utilizar la combinación DATA-READ:

1. Puede utilizar tantos elementos de DATAS como necesite en el programa, pero tenga en cuenta que la lista de elementos de la DATA no ha de sobrepasar una línea de programa. Esto es, no se admiten más de 96 caracteres por línea, incluyendo el número de línea y la palabra DATA. Cuando tenga más datos que utilizar, basta con empezar una línea nueva (número de línea, DATA, elemento, elemento, etc...).

2. Cuando hayan de leerse (READ) los datos (DATA), asegúrese de que la variable en la que se va a almacenar el dato sea del mismo tipo. Números han de guardarse en variables numéricas (hay una excepción que veremos más adelante), y cadenas han de almacenarse en variables de cadena.

De manera que la siguiente lista de datos:
1Ø DATA 1Ø, 12, 15, 7, 9Ø. 16, 122, 9999
han de ser leídos y almacenados en variables del tipo A, AA, XY, C1, H5, etc.

Mientras que una lista de datos como:
2Ø DATA "LONDRES", "NUEVA YORK", "SYDNEY", "SINGAPUR"
solamente se pueden almacenar en variables del tipo A$, B$. etc.,

NOTA: una variable de cadena es una variable alfanumérica. Aceptará números en una lista de datos alfanumérica (siempre y cuando permanezcan encerrados entre comillas). El ordenador los tratará como cadenas, con el problema de que no podrá hacer ninguan operación aritmética con ellos. Pero veremos algo más so-

bre ello a continuación, cuando tratemos la instrucción VAL.

## RESTORE

Esta instrucción se utiliza juntamente con los comandos READ y DATA. Cuando una lista de datos, que ya han sido leídos, han de utilizarse de nuevo, no se podrán releer a no ser que hagamos un RESTORE.

Vuelva a nuestro programa ejemplo y sustituya la línea 5Ø STOP, por 5Ø RESTORE: GO TO 1Ø. Ejecute el programa. Con asombrosa velocidad verá el mismo juego de números (mayores de 25) repetirse una y otra vez.

El efecto de RESTORE es que se hace un reset del puntero de la DATA posicionándolo de nuevo al comienzo de la lista de los items de la DATA. Cuando hacemos referencia a la lista de items queremos decir la totalidad de los items. o elementos.

## DEF FN

Esta instrucción significa DEFinición de FuNción y es una de las herramientas más potentes que el BASIC del ZX Spectrum puede ofrecer, particularmente en el área de procesos complicados de cálculo. En realidad, puede sustituir en una línea de programación a una subrutina completa, y se pueden realizar funciones extras.

Tome por ejemplo, la función INT, que significa INTeger (entero). Con esta función el ordenador visualizará —o contestará, como dicen los programadores— la parte entera de un número real. Teclee lo siguiente:

LET X = INT(45.6789) y pulse ENTER
PRINT X y pulse ENTER

El ordenador le contestará con el valor 45, que es la parte entera de 45.6789. Sin embargo, hay suficientes situaciones donde el usuario requiere la parte decimal y no la entera. Esto se obtiene de la forma siguiente:

LET X = 45.6789 y pulse ENTER
LET X = X — INT(X) y pulse ENTER

El ordenador contestará con .6789 que es la parte fraccionaria del número.

En principio la línea X = X — INT(X) puede ser tratada como una subrutina, pudiéndola situar en cualquier parte del programa. Aunque no siempre es práctico. Por-

que cada vez que quiera hallar la parte fraccionaria de un número, esta variable habrá que sustituirla por X, y entonces el programa puede dirigirse a la subrutina, volver y sustituir de nuevo la variable X. Este es un poco engorroso y se puede evitar mediante el uso de DEF FN.

El formato del comando DEF FN es el siguiente:

DEF FN nombre de variable (una letra) = (a continuación la fórmula con las variables que definen la función).

Nombre de variable de una letra significa que la función ha de tener un nombre, definido por una letra. El ordenador no aceptará nombres de dos letras en una instrucción DEF FN. Esto por consiguiente, limita el número de funciones definidas a 26 que es el número de letras del alfabeto.

En cuanto a la variable, puede tomar cualquier variable porque, cuando se utiliza la nueva función en el programa, esta variable se sustituirá por la que usaremos para resolver el problema. Una función que dé el valor fraccionario de un número puede ser algo parecido a lo siguiente:

1Ø DEF FN F(X) = X — INT(X)

Un programa pequeño lo demostrará. Teclee NEW y pulse ENTER primero (si tiene otro programa en el ordenador), a continuación teclee:

```
10 DEF FN F(X)=X-INT (X)
20 INPUT A
30 LET A=FN F(A)
40 RPINT A
```

Ahora ejecute el programa y cuando el ordenador le pregunte por los datos de entrada, teclee por ejemplo, el número, 123.456789. Después de haber tecleado el número, el ordenador le responderá con lo siguiente: Ø.456789Ø2. Que, por supuesto, no es la fracción del número que usted tecleó. Esto se debe a que el ordenador tiene problemas con la exactitud. Por ejemplo, si teclea 45.6 el ordenador le devolverá Ø.5999999.

A continuación le presentamos un pequeño truco que resolverá el problema. Amplíe la línea 1Ø con lo siguiente:

1Ø DEF FN F(X) = (X — INT (X)) + 1.5E-9)

El número 1.5E-9 es una cantidad muy, muy pequeña, que añadida al resultado de X — INT (X) ayudará a conseguir la fracción correcta.

## IMPORTANTE

Observe que esta función no responde con las cantidades correctas cuando trabaja con números negativos (número precedido del signo menos). Esto se puede solucionar de la siguiente forma:

1Ø DEF FN F (X) = SGN(X) * ((ABS(X) — INT (ABS(X)) + 1.5E-9)

El cómo y porqué de esta función es algo que tendrá que resolver por su cuenta. Estudie el funcionamiento de la instrucción SGN(X) y ABS(X). Intente establecer la utilidad de todos los paréntesis usados en la expresión. Recuerde, que uno de los aspectos más emocionantes de la programación es que tiene que depender de su propia inventiva en muchos casos. El mejor programador de ordenadores es aquel que no ha aprendido todo por los libros.

Observe los dos casos que acaba de comprobar en el programa anterior:

1. La instrucción de DEF FN se encuentra al principio del programa. ESTO ES OBLIGATORIO. Antes de que cualquier programa que utilize DEF FN pueda funcionar correctamente, han de definirse esas funciones antes de que se puedan aplicar en cualquier otra parte del programa. Es obvio que ha de hacerse en la rutina de inicialización del programa.

2. Vea la línea 3Ø. La variable X de F(X) ha sido sustituida por la variable A. Esto le convencerá de que cada variable dentro del contexto de una instrucción DEF FN funcionará de la misma manera. De manera que cuando desee saber la parte fraccionaria de cualquier otra variable, haga los siguiente:

LET B = FN F(B), o LET B = FN F(K), o también puede hacer LET B = FN F(C), etc.

## DIM

DIM significa DIMensión y de nuevo nos encontramos con una herramienta muy potente del BASIC, pero desafortunadamente es una de las instrucciones menos comprendidas. Por esta razón nos extenderemos más de lo habitual describiendo el correcto funcionamiento de ésta, considerando que el manual del ZX Spectrum podría haber sido un poco más claro en este aspecto.

Para ello explicaremos lo que es una variable subindicada y arrays (tablas). Como puede ver, estos dos

conceptos son similares. La idea de una variable subindicada es que tienen algo en común. Para ilustrar esto, volvamos al ejemplo del archivo (ver el capítulo II). Tire del fichero (las variables) A. Imagine que en este fichero A nos encontramos con un conjunto de ficheros encadenados, cada uno de los cuales contiene datos o información que tienen algo en común, particularmente que todos están almacenados en el fichero A (o variable A). Puede dar a cada uno de estos ficheros un nombre, como por ejemplo A(1), A(2), A(3),...

Esto es lo que ocurre exactamente en el BASIC: una cadena de datos se pueden almacenar en una tabla de variables subindicadas. El formato de una variable subindicada es A(X), donde la X puede ser cualquier valor entre Ø y 65545. De aquí que A(1), B(1Ø), C(23) sean todas variables subindicadas válidas. Pero tenga en cuenta que la variable A(1) NO es lo mismo que la variable A1.

¿Pero qué tiene que ver DIM con todo esto? Pues sencillamente eso —TODO. DIM es crucial aquí. porque determina el número de items de datos que una variable subindicada puede manejar. La tabla (array) se ha de DIMENSIONAR primero. Pruebe esto:

LET A(12) = 45 y pulse ENTER

El ordenador le responderá con un menseaje de error: «Variable not found» («Variable no encontrada»). Esto se debe principalmente a que no ha DIMensionado esta variable subindicada. Ahora teclee esto otro:

DIM A(2Ø):LET A(15)= 92:PRINT A(15) y pulse ENTER

El ordenador le contestará con el valor 92. O sea que no tendrá problemas, porque la variable subindicada ha sido DIMensionada correctamente. Pero de nuevo, cuando se exceda en la dimensión de, en esta caso 2Ø, el ordenador le responderá con un mensaje de error, generalmente con «suscript wrong» (índice equivocado).

Hasta aquí hemos discutido las variables numéricas subindicadas, ahora tenga en cuenta que, por ejemplo A(15) contiene números en vez de cadenas.

Sin embargo, también hay cadenas, con subíndices que a su vez lo son también. El DIMensionamiento de dichas tablas es un poco más complicado. Porque la instrucción DIM no sólo ha de indicar el tamaño de la variable (el número de items que desea almacenar), sino que también tendrá que especificar la longitud (número de caracteres) de datos. Vea lo siguiente:

DIM A$ (12,10)

El 12 indica el tamaño de la tabla, habrá 12 elementos (o items) almacenados en ella. El 1Ø por otro lado indica la longitud (número de caracteres) de esos datos. Para demostrar esto teclee y ejecute el siguiente programa:

```
  10 DIM A$(12,10)
  20 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL"
  30 DATA "MAYO","JUNIO","JULIO"
,"AGOSTO"
  40 DATA "SEPTIEMBRE","OCTUBRE"
,"NOVIEMBRE","DICEMBRE"
  50 LET I=1
  60 READ A$(I)
  70 PRINT A$(I)
  80 LET I=I+1
  90 IF I=13 THEN GO TO 200
 100 GO TO 60
 200 PRINT
 210 PRINT "READY"
```

Este programa es autoexplicativo. Ahora experimente un poco con la línea 1Ø. Por ejemplo sustitúyala por: 1Ø DIM A$(12,5) y ejecute el programa de nuevo.

Este pequeño programa se caracteriza porque muestra la utilidad de las cadenas; ofrece la oportunidad de almacenar datos de la misma naturaleza. Por ejemplo, puede almacenar los nombre de las ciudades en cadenas, como las siguientes:
A$ = "AMSTERDAN", L$ = "LONDRES", B$ = "BERLIN", etc, pero nos encontramos con dos problemas.

1. Llevaría mucha programación el acceder a estos nombres, y

2. Se quedaría sin variables muy pronto al haber solamente 26 letras en el alfabeto.

Pero cuando se dé cuenta de que en un principio una cadena puede almacenar más de 6ØØØØ elementos, estaremos ante algo totalmente diferente.

Hay algo más que deberá saber de la instrucción DIM. Hasta el momento hemos visto los denominados arrays uni-dimensionales (vectores), por ejemplo DIM A (2Ø). De todos modos se pueden dimensionar de dos, tres y hasta más si se desea. Ejemplo: DIM A(2Ø,5), DIM B(2Ø,5,12).

NOTA: no confunda esto con DIN A$(2Ø,5) (observe el símbolo de la cadena $); DIM A(2Ø,5) es algo totalmente distinto. Sin embargo, éste es un punto en el que le aconsejamos no se preocupe demasiado a estas alturas de su «carrera» informática. Los arrays multidimensionales son para programadores más avanzados y por consiguiente no entran en el ámbito de este libro.

## IMPORTANTE

Es una buena práctica de programación que todas las intrucciones DIM se sitúen al comienzo del programa.

## STOP

En contraste con otros ordenadores, el ZX Spectrum no tiene la instrucción END. Es una pena porque aunque la sentencia END es opcional en las versiones del BASIC, podría resultar muy útil. END es el fin físico de un programa. Dondequiera que se ponga en un programa, acabará de inmediato. Esto es necesario en programación estructurada donde un programa ha de finalizar en la rutina principal.

Ahora sólo puede utilizar el comando STOP, el cual a resultas del uso aparecerá un mensaje al final de la pantalla. Esto no es muy elegante. Una solución más elegante sería sustituir la instrucción STOP al final de la rutina principal por un GO TO, que dirigirá el puntero del programa al final del programa.

La última línea podría ser:

PRINT:PRINT:PRINT"LISTO", o

PRINT:PRINT:PRINT"FIN DEL PROGRAMA".

Siendo la última línea, el programa finalizará.

En lo que se refiere al comando STOP en sí, puede utilizarlo para parar un programa momentáneamente dándole la oportunidad de analizar los contenidos de las variables. Esto es particularmente importante en programas matemáticos que se estén desarrollando. Ya que el comando STOP será útil durante el proceso de «debugging» (corregir los errores).

# CAPITULO 7:
# Más herramientas del BASIC

## FOR-TO-NEXT-STEP

Les presentaremos un juego de instrucciones esenciales del BASIC. Esenciales porque permiten construir bucles de una forma única en BASIC. En capítulos anteriores hemos visto algunas de las técnicas de los bucles donde probamos y experimentamos con las instrucciones IF y GO TO. Sin embargo, la combinación FOR-TO-NEXT-STEP, es mucho más versátil y muestra en realidad la flexibilidad del BASIC.

El formato (o la sintaxis) es:

FOR (nombre de variable) = (valor inicial) TO (valor final) STEP (incremento).

Por ejemplo:

```
1Ø FOR I = 1 TO 1ØØ STEP 2
```

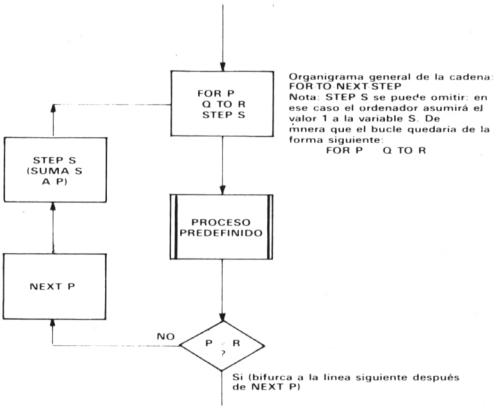
Antes de continuar veamos el siguiente ejemplo:

```
1Ø FOR I = 1 TO 1ØØ
2Ø PRINT I;
3Ø NEXT I
```

Teclee el programa y ejecútelo.

Lo que ocurrirá es que el programa hará aparecer en pantalla y horizontalmente, los números del 1 al 1ØØ y luego se parará. ¿Cómo funciona?

En la línea 1Ø el programador determina que el contenido de la variable I sea incrementado hasta 1ØØ continuamente. La línea 2Ø imprime el contenido actual de la variable I. La línea 3Ø comprueba el contenido de la variable I, y si su valor no es 1ØØ, entonces lo incrementará en una unidad y el flujo del programa retorna a la línea 1Ø que es el inicio de la instrucción FOR-TO-NEXT. Esto continúa hasta el momento en que el test de la línea 3Ø sea cierto, o sea, que I valga 1ØØ. En este caso el programa no vuelve al principio de las instrucciones del bucle, sino que ejecutará a partir de la línea 3Ø. Para un correcto entendimiento de esta herramienta de programación estudie el siguiente organigrama detenidamente.




Organigrama general de la cadena: FOR TO NEXT STEP
Nota: STEP S se puede omitir: en ese caso el ordenador asumirá el valor 1 a la variable S. De mnera que el bucle quedaría de la forma siguiente:
FOR P Q TO R

## Flujo del programa en palabras

- Inicializar el bucle FOR-NEXT: la variable es igual a Q (valor inicial) hasta R (valor final).
- Ejecutar el proceso (proceso predefinido).
- Test P: IF, si P es igual a R entonces bifurca fuera del bucle FOR-NEXT.

  IF, si P no es igual a R, entonces vuelve al principio del bucle, y suma S a la variable P.
- Reinicializar el ciclo.

La mejor forma de ilustrar un bucle FOR-NEXT es probando un programa. Teclee lo siguiente:

```
  10 DIM A(20): REM DIMENSIONA L
A MATRIZ
  20 DATA 1,5,12,3,0,23,55,46,24
,34,22,7,9,33,29,94,4,8,10,11
  30 FOR I=1 TO 20
  40 READ A(I)
  50 NEXT I
  60 REM ESTE ERA EL PRIMER BUCL
E FOR-NEXT, A CONTINUACION HAY O
TRO
  70 FOR J=1 TO 20
  80 PRINT A(J)
  90 NEXT J
```

Ejecute el programa y compruebe lo que ocurre. En realidad visualiza todos los números de la lista de la DATA de la línea 2Ø, gracias a los dos bucles FOR-NEXT.

Lo que ocurre es que el primer bucle comprende las líneas 3Ø, 4Ø y 5Ø. La línea 3Ø nos indica que la va-

riable «I» hay que incrementarla en una unidad en cada ciclo durante un total de 2Ø ciclos que se ejecutará este bucle. (FOR I = 1 TO 2Ø). De manera que en el primer ciclo I es igual a 1. En la línea 40 el programa encuentra la primera variable subindicada, que efectivamente será A (I) o A (1). La instrucción leerá el primer elemento de la DATA y lo almacenará en la variable A (1). Por casualidad el primer valor de la sentencia DATA es 1 (vea la lista en la línea 2Ø). El programa se dirigirá hacia la línea 5Ø, que dice NEXT I. En esta línea tiene lugar un test: ¿I = 2Ø? Si la respuesta es no, el valor de 1 se le añade a I, y el contador de programa vuelve a la línea 3Ø, y ejecuta de nuevo el proceso enmarcado en las líneas 3Ø, 4Ø y 5Ø. Después de 2Ø ciclos (FOR I = 1 TO 2Ø), el valor de I se habrá incrementado hasta 21, entonces el test de la línea 5Ø (NEXT I) se comprueba que es cierto y el programa se bifurca fuera del bucle.

De esta forma, con el uso de FOR-NEXT como herramienta importante, la lista completa de elementos de DATA se pueden leer hacia unas variables. Por lo que, en casos como estos, las variables subindicadas A (1) y A (2Ø) se rellenan con estos datos, uno cada vez. Después de que el programa bifurque más allá del bucle de la línea 5Ø, va a la línea 7Ø (la 6Ø se ignora porque es una instrucción REM), donde se encontrará con otro bucle FOR-NEXT. El objeto es imprimir en la pantalla el contenido de cada variable subindicada de la cadena A ( ). Ahora la variable a incrementar y a probar será la J, y el valor del incremento va desde 1 hasta 2Ø, al igual que con I. De manera que este ciclo que comienza en la línea 7Ø, imprime el valor de la variable subindicada A (J), incrementa el valor de J en una unidad en la línea 9Ø mientras que el puntero del programa se dirige a la línea 7Ø. El ciclo termina cuando la J vale 21. Todos los valores son visualizados de esta forma por pantalla.

Y ahora un pequeño ejercicio. Sustituya la línea 7Ø por

```
7Ø FOR J = 1 TO 2Ø STEP 2
```

y ejecute el programa de nuevo. Ahora sólo verá aparecer diez números en la pantalla. Y, lo que es más, los números extraídos de la lista de elementos de las DATAS aparecen alternados. El primero, el tercero, el quinto, y así sucesivamente.

De esta forma solamente once números, de las variables con subíndice impar aparecen en pantalla. El or-

denador escribe el contenido de A (1), A (3), A (5), etc. Como observará, hay un incremento de 2 unidades (STEP 2) entre dos subíndices.

## IMPORTANTE

El bucle FOR-NEXT también puede trabajar del modo contrario, esto es, disminuyendo sucesivamente. Sustituya la línea 7Ø por:

7Ø FOR J = 2Ø TO 1 STEP —1

Ejecute el programa de nuevo y observe cómo se van imprimiendo los contenidos de las variables A (2Ø), A (19), A (18), etc.

NOTA: En este caso es obligatorio poner STEP —1, cuando quiera hacer decrecer el bucle en una unidad por ciclo. Como sabrá, si lo que quiere es incrementarlo, la instrucción STEP será opcional. Lo contrario no es cierto.

Cuando esté de lleno en el mundo de la programación, apreciará esta particular característica del BASIC más que cualquier otra cosa, ya que efectúa una diversa gama de trabajos, como procedimientos repetitivos de alto nivel, que suele ser precisamente el trabajo principal para el que se ha hecho el ordenador.

Queremos resaltar que esta caracterítica es particularmente útil a la hora de escribir lo que se denominan «bucles anidados». Le aconsejamos que lo proponga para cuando haya avanzado considerablemente su nivel de programación. Sobre los bucles anidados, puede leer en cualquier libro de programación en BASIC, pero es un trabajo un poco arriesgado, por lo que no entra dentro del esquema de nuestro libro.

## INKEY$

Es un comando bastante peculiar al principio, pero muy útil cuando se entiende correctamente. ¿Qué es lo que hace? El nombre lo aclara: KEY IN (teclea una cadena) $tring. Sin embargo, la cadena no la podemos tomar literalmente, porque el comando no quiere que le tecleen cadenas de datos, sólo admite un caracter. Según el programa, este comando aceptará un caracter específico del teclado y actuará en consecuencia. El siguiente programa de demostración aclarará este paso:

```
  10 CLS : REM limpia pantalla
  20 PRINT "SOCORRO!!"
  30 PRINT "HE ENTRADO EN UN "
  40 PRINT "CIRCULO VICIOSO"
  50 PRINT "AYUDAME PULSANDO UNA
 TECLA"
  60 PRINT "LO SIENTO...NO SE CU
AL ES"
  70 PRINT : REM linea en blanco
  80 IF INKEY$<>"G" THEN GO TO 8
0
  90 PRINT "UN MONTON DE GRACIAS
"
 100 PRINT "ESTOY FRITO...."
```

Ejecútelo. Verá cómo solamente el texto programado en las líneas 2Ø y 6Ø aparece en pantalla. Podrá pulsar cualquier tecla y no ocurrirá nada, hasta que pulse la tecla G. Entonces las líneas 9Ø y 1ØØ son las que ejecutan a continuación y el programa finaliza.

La pista está en la línea 8Ø. El programa ejecuta hasta la línea 8Ø, y allí se encierra en un bucle sin fin. La instrucción INKEY$ comprueba continuamente si se ha pulsado una tecla y, si es así, comprueba si ha sido la tecla G. Mientras que la tecla pulsada no sea la G, el programa continuará en un bucle hasta la línea 8Ø. Pero en cuanto se pulsa la tecla G, entonces la condición se verifica y el programa se sale del bucle y ejecuta la línea siguiente del programa, en este caso la línea 9Ø.

NOTA: cuando teclee este programa, tenga en cuenta que el operador relacional < > tiene que teclearse por medio de la tecla que tenga ese caracter particular. Si teclea <, seguido de >, entonces le saldrá algo parecido a lo que se desea pero ¡NO FUNCIONARA! El Spectrum reaccionará con un signo de interrogación y un cursor parpadeante.

El comando INKEY$ es muy útil para hacer unos buenos menús por pantalla. Para probarlo, vuelva al gran programa de conversiones. Vea las líneas 76Ø y 82Ø. Sustitúyalas por las siguientes:

```
 760 IF INKEY$="" THEN GO TO 760
 790 IF INKEY$="1" THEN GO TO 96
0
 800 IF INKEY$="2" THEN GO TO 11
30
 810 IF INKEY$="0" THEN GO TO 87
0
```

Ejecute esta nueva variación del programa. El menú aparece en la pantalla, pero no encontrará el cursor destellante. Sin embargo, cuando pulse una de las teclas de las opciones (1, 2, Ø), el programa reaccionará inmediatamente. No tiene que pulsar la tecla ENTER.

Este es un método extremadamente eficiente para dirigir el flujo de un programa hacia otro lugar del programa. Lo que ocurre en la línea sustituida 76Ø es lo siguiente:

En la línea 76Ø el ordenador espera un caracter del teclado. Mientras no haya caracter alguno desde el teclado, INKEY$ no contendrá nada (indicado por = ""). En este caso la línea 76Ø es un bucle sin fin, que sólo se interrumpirá cuando el caracter probado sea un 1, 2 ó Ø.

Ahora tiene experiencia con la utilidad del comando INKEY$; igualmente puede alterar el resto de las opciones del programa de conversión. Vea las líneas 1.Ø4Ø-1Ø6Ø, 121Ø-123Ø, 287Ø-289Ø y 3155-317Ø, Inténtelo, será un ejercicio muy útil.

## EL COMANDO LET

Ya ha utilizado este comando extensamente y conoce sus funciones, inicializa las variables. Por ejemplo, LET S = B + C. Lo que usted no sabrá es que el Spectrum es uno de los pocos ordenadores que hacen que la instrucción sea obligatoria. En la mayoría de los dialectos del BASIC, del comando LET es opcional. Lo que significa que está permitido ponerlo y no tan permitido escribirlo. Algunos BASIC entienden lo mismo con LET A = B + C que con A = B + C.

## STR$ (X)

Esta función transforma una variable numérica en una alfanumérica o cadena. A veces es más fácil manejar cadenas numéricas que variables numéricas, especialmente cuando hay que imprimir cifras mediante tablas (filas y columnas). La idea es la siguiente:

```
10 LET X=123.456: LET Y=23.500:
LET Z=10225.009
20 PRINT X
30 PRINT Y
40 PRINT Z
```

Este programa convertirá el valor de X en una cadena de números. Hay situaciones en que esto puede ser muy útil, como ya indicamos, cuando se quiere imprimir una tabla de cifras por columnas. En BASIC, a veces es difícil manejar las variables numéricas. Por ejemplo, si quiere imprimir los siguientes números por columnas:

```
    123.456
     23.5ØØ
 1Ø225.ØØ9
```

No piense que se puede hacer fácilmente de la siguiente forma:

```
10 LET X=123.456: LET Y=23.500
LET Z=10225.009
20 PRINT X
30 PRINT Y
40 PRINT Z
```

El resultado será el siguiente:

```
123.456
23.5
1Ø225.ØØ9
```

Sin embargo, cuando convierta esos números a cadenas entonces podrá escribir sus programas de manera que la X se convierta en X$, la Y en Y$, la Z en Z$. En el primer caso tiene que añadir las cadenas siguientes a X$: X$ = "  " + X$ (esto añade dos espacios en blanco a la izquierda del número), Y$ = X$ "  " + Y$ + "ØØ" (esto añade tres espacios en blanco a la izquierda del número y dos ceros a la derecha). Entonces cuando imprima las cadenas obtendrá una columna perfecta:

```
    123.456
     23.5ØØ
 1Ø225.ØØ9
```

## VAL (X$)

Cuando los números se almacenan en cadenas, ya no sirven para poder hacer operaciones aritméticas con ellos. Por ejemplo, cuando A$ contenga un 1Ø y B$ un 2Ø, no podrá hacer una operación como la siguiente:

LET C$ = A$ + B$

y espere algún resultado en C$ (3Ø). Al contrario, C$ almacenará el valor "1Ø2Ø" (en este caso las dos cadenas se han unido). La multiplicación, la división y la

resta, estan totalmente fuera de lugar.

Será otra cuestión cuando utilize la función VAL (X$). Esta función nos da el valor numérico de una cadena. De manera que cuando haga:

LET C = VAL (A$) + VAL (B$)

entonces obtendrá el valor 30 en la variable C.

## UTILICE SU IMAGINACION

Como ha podido comprobar, hemos dejado de lado algunas funciones numéricas. Sin embargo, si le gustan las matemáticas, esas funciones tendrán ya algún significado para usted y no tendrá excesivos problemas en conocer como trabajan. En un apéndice separado encontrará algunas funciones útiles (creadas con la instrucción DEF FN) que le harán más fáciles los cálculos trigonométricos y similares.

Este libro le ha iniciado en la programación del BASIC del Spectrum. Cuando haya hecho todos los ejercicios, fáciles y difíciles, habrá adquirido la suficiente habilidad para no necesitar más explicaciones. Encontrará respuestas a los problemas. Está listo para «abrir sus alas», por lo que nuestro consejo es que «vuele y explore el mundo de la programación» en BASIC por su cuenta. En cuanto al ZX Spectrum, sabrá que lo puede utilizar para algo más que para problemas aritméticos, tratamiento de textos, etc. El ZX Spectrum le ofrece color, gráficos y sonido.

Un último consejo. No se precipite. Primero familiaricese con lo básico del BASIC, antes de profundizar en el color, sonido y los gráficos. Pero cuando se encuentre preparado, el manual del Spectrum y muchos libros más que hay en el mercado ofrecen una guía inmejorable.

Le deseamos mucha suerte y que se divierta.

# APENDICE A:

## Algunas funciones matemáticas

## con formato DEF FN

### ENTEROS Y FRACCIONES

El ZX Spectrum ya tiene definida la función INT(X), que calcula la parte entera de un número real. Sin embargo, no siempre el valor hallado es el correcto. Por ejemplo, si desea hallar la parte entera de un número negativo, puede acabar con algo totalmente imprevisto. Pruebe esto:

LET A = —5.9: PRINT INT(A)

El ordenador imprimirá —6, en vez de —5, (como debe ser).

La siguiente función definida le ayudará:

DEF FN I(X) = SGN(X) * INT(ABS(X))

Sitúe esta función al comienzo de un programa y cada vez que desea hallar la parte entera de un número negativo, lo podrá calcular gracias a esta función.

Ejemplo:

```
 10 DEF FN I(X)=SGN (X)*INT (AB
S (X))
 20 INPUT A
 30 LET A=FN I(A)
 40 PRINT A
```

Cuando un programa utiliza la parte entera de un número real, puede ser igual de útil el poder acceder a la parte fraccionaria. A continuación le exponemos una función definida que puede ser utilizada en combinación con la función de la parte entera.

DEF FN F(X) = SGN(X) * ((ABS(X) — FN I(ABS(X))) + 1.5E—9)

(Esta función es similar a la descrita en el capítulo 7).

## HORAS Y GRADOS, MINUTOS Y SEGUNDOS A HORAS DECIMALES O GRADOS

Esta función es muy larga —tecléela con cuidado.

DEF FN D(X) = SGN (X) * (INT(ABS(X)) + INT(1ØØ* FN F(ABS(X)))/6Ø + FN F (1ØØ * ABS(X))/36)

Observe que la función de hallar la parte fraccionaria se ha aplicado en ésta. ¿Cómo utilizarla?

```
  3 REM Poner aqui la DEF FN de
la funcion entera
  4 REM Poner aqui la DEF FN de
la parte fraccionaria
  5 REM Poner aqui la DEF FN de
la funcion grados
 10 INPUT "HH.MMSS";HS
 20 LET HS=FN D(HS)
 30 PRINT HS
```

Tenga en cuenta que el formato de entrada es HH.MMSS. Por ejemplo, si quiere convertir 15 hr. 33 min. 17 seg. tecléelo de la forma siguiente: 15.3317. Cuando lo convierta con el programa anterior debe obtener el resultado siguiente: 15.5547222

A continuación expondremos la función que convierte horas decimales o grados a horas (o grados) y minutos (solamente).

DEF FN S(X) = FN I(X) + FN F(X) * Ø.6

Observe de nuevo que en esta función se han aplicado otras dos funciones (entera y fracción). Tenga en cuenta que en casos como éste, donde las funciones definidas hay que incorporarlas a otras funciones definidas, siempre tendrá que definir en primer lugar la función que luego va a pertenecer a otra.

## FUNCION DE REDONDEO

El resultado de expresiones aritméticas, en la mayoría de los casos, da unos resultados de muchos dígitos después del punto decimal. Esto no siempre interesa. A continuación exponemos una función que redondeará a dos dígitos después del punto decimal, mientras que

esta parte fraccionaria a su vez se redondea.

DEF FN R(X) = FN I(X * 1ØØ) + Ø.5)/1ØØ

Esta función redondeará un número como 45.67891 a otro que será 45.68.

## TRIGONOMETRIA

El Spectrum ofrece las funciones trigonométricas SIN, COS, TAN, ASN (arco seno), ACS (arco coseno), ATN (arco tangente). En comparación con otros ordenadores que sólo tienen SIN, COS, TAN y ATN su ZX Spectrum es una máquina mejor en lo que se refiere a funciones trigonométricas. Pero aún existe un pequeño problema, estas funciones trabajan en radianes. Muchas personas prefieren trabajar con grados antes que con radianes. Para solucionar el problema, puede crear algunas funciones que lo resuelvan adecuadamente.

Para convertir radianes a grados tiene que utilizar la famosa constante PI que en su Spectrum tiene tecla propia. El seno, coseno y tangente las ponemos en el siguiente formato de función definida:

DEF FN A(X) = SIN (X * PI/18Ø)

El arco seno, arco coseno y arco tangente corren la misma suerte siendo sus funciones derivadas de la siguiente:

DEF FN B(X) = ASN(X) * 18Ø/PI

Un ejemplo:

```
 5 DEF FN C(X)=COS (X*PI/180)
10 INPUT A
20 LET A=FN C(A)
30 PRINT A
```

En la línea 1Ø tecleará el dato de entrada en grados y en la línea 3Ø el ordenador responderá con el resultado correcto.

# APENDICE B:

Una solución al ejercicio propuesto en el capítulo 2:

```
  10 REM ESTE PROGRAMA CONVIERTE
 LIBRAS, STONES, ONZAS Y GRAINS
  20 REM EN GRAMOS
  30 INPUT "TECLEE EL NUMERO DE
LIBRAS ";LB
  40 LET GM=LB*453.6: REM CONVIE
RTE LIBRAS EN GRAMOS
  50 LET SE=14*GM: REM CONVIERTE
 STONES EN GRAMOS
  60 LET OU=GM/16: REM CONVIERTE
 ONZAS EN GRAMOS
  70 LET GR=GM/16/16: REM CONVIE
RTE GRAINS EN GRAMOS
  80 PRINT "STONES A GRAMOS ";SE
  90 PRINT "LIBRAS A GRAMOS ";GM
 100 PRINT " ONZAS A GRAMOS ";OU
 110 PRINT "GRAINS A GRAMOS ";GR
```

Otros libros de interés:

- LAS PRIMERAS 15 HORAS CON ZX SPECTRUM
  por
  Derek Ellershaw
  y
  Peter Schofield

- LOS MEJORES PROGRAMAS PARA EL SPECTRUM
  por
  Robert Erskine, Humphrey Walwyn
  con
  Paul Stanley, Michael Bews

Cuando el Spectrum y su dueño ya son amigos, cuando uno empieza a cansarse de jugar a marcianitos, ¿cómo seguir adelante en el disfrute del ordenador?; ¿es necesario digerir alguno de esos engorrosos tratados de programación? A quienes se han planteado estas preguntas está dedicado este libro, escrito en estilo ameno y muy práctico.

El temario incluye:

- Reglas y herramientas del BASIC.
- La técnica de los organigramas.
- Cómo planificar un programa.
- El fabuloso mundo de las rutinas.
- Variables y cadenas.
- Algunas funciones matemáticas usuales.

Un libro fundamental en la biblioteca de los aficionados al Spectrum, para que ellos mismos se ayuden a sacar todo el partido posible de esta máquina y de los ordenadores que usarán en el futuro.

EDITORIAL PARANINFO S.A
Magallanes, 25 - 28015 Madrid

Y152
El Corte Inglés
Talla X>>>750
P.V.P.